Project Network Results Format
ESA Contract No. 10596/93/NL/FG Rider No. 2

STEP Application Protocol

STEP-NRF – Network-model Results Format

NRF-002-AP

Version 1

Release 1.4 (15 November 1996)

European Space Agency

French Space Agency

Fokker Space

Association GOSET

ESPRI CONCEPT

Abstract

This document is a STEP based application protocol for the electronic exchange of bulk results data associated with network models. A network model in this context is a generic representation of an engineering object (or a collection of related engineering objects) by a collection of discrete network nodes and relationships between these nodes. The bulk results are characteristic, predicted, assigned or observed properties for components of the engineering object(s), which are sampled during an analysis, test or operation run.

The protocol may also be called NRF-AP, for short.

STEP (STandard for the Exchange of Product model data) is the casual name for the ISO 10303 standard. The NRF-AP is a so-called External Application Protocol, which means it is not an official ISO work item. Its development is initiated and funded by the European Space Agency (ESA) in co-operation with the French Space Agency (CNES).

ISO 10303 / STEP is a series of International Standards for the computer-interpretable representation and exchange of product data. The objective is to provide a neutral mechanism capable of describing product data throughout the life cycle of a product, independent from any particular system. The nature of this description makes it suitable not only for neutral file exchange, but also a basis for implementing and sharing product databases and archiving. The development of the ISO 10303 standard is a world-wide undertaking covering all engineering disciplines and all industrial branches. It started in 1984 and is the most comprehensive effort in its kind.

Current status and ongoing activities

The current Release 1.4 is the fifth release for Version 1 of the Network-model Results Format Application Protocol.

This release is the first complete Application Protocol including an implementable Application Interpreted Model (AIM) that covers all aspects of bulk results exchange for network models. This release forms the basis upon which the first version of the NRF read/write library will be developed.

Colophon

This document was produced as part of the ESA project “Network Results Format” (ESA/ESTEC Contract Number 10596/93/NL/FG Rider No. 2).

The project team consisted of:

- Pau Planas Almazan, ESA/ESTEC/YCV, Noordwijk, The Netherlands, pau@yc.estec.esa.nl.
  ESA project manager and reviewer.
- Hans Peter de Koning, Fokker Space BV, Leiden, The Netherlands, h.p.de.koning@fokkerspace.nl.
  Design leader, co-author and reviewer.
- Hans de Wolf, Fokker Space BV, Leiden, The Netherlands, h.de.wolf@fokkerspace.nl.
  Author of all clauses / annexes except Clause 5 and Annexes A, B, C, H, J.
- Pascal Huau, Association GOSET, Nanterre (Paris), France.
  Reviewer and author of Clause 5 and Annexes A, B, C, H, J.
- Dominique Molin, Association GOSET, Nanterre (Paris), France.
  Reviewer and co-author of Clause 4 and Annex G.
- Eric Lebegue, ESPRI Concept, Sophia Antipolis, France.
  Reviewer and responsible for NRF read/write software library.

The review team consisted of:

- Jean-Luc Le Gal, CNES, Toulouse, France – Reviewer.
- Christian Puillet, CNES, Toulouse, France – Reviewer.
- Jean-Philippe Goube, Intespace, Toulouse, France – Reviewer.

The document was prepared with the FrameMaker®4 document processor. Graphics were produced using Deneba Canvas 3.5 and Adobe Photoshop 3.0.

Document Status Sheet

Document Status Sheet for NRF-002-AP

version, release, date	*reason for change*
v1, r1.0, 24-Jun-96	First release. Created using the [STEP-TAS-AP] as a template.
v1, r1.1, 19-Jul-96	Changed name from *Network Model Bulk Results* (NMBR) to *Network-model Results Format* . Acronym remained NRF. Added the **product_structure** UoF. Added the **parameterised_functions** UoF. In **date_and_time** UoF: • Declared the [STEP-41] objects **ordinal_date**, **day_of_week** and **week_of_year** out of scope. In **generic_network_model_representation**: • Changed **network_model_component** EXPRESS code. • Changed definition of **generic_network_model**. • Added class attribute to **network_node** and **network_node_relationship** • Added examples to **network_node**. • Specified purpose of **network_node_usage**. • Added **network_node_relationship_usage**. In the **bulk_results** UoF: • Added *node_classes* and *nrelationship_classes* attributes to **run_context**. • Integrated **descriptive_domain_list** into **descriptive_value_list**, removed **enumerated_list** . • Removed **model_component_class**. • Extended description of **property_class**. • Changed **qualified_property_class** into **property_class_with_role**. • Added **property_class_role** type**.** • Redesign of tensor properties, replacing properties with tensor nature by tensors of properties. Corrected minor text errors.
v1, r1.2, 14-Aug-96	Added **sensor_represensentation** UoF. Added **sensor_oriented_results** UoF. In **date_and_time** UoF : • added [STEP-41] **date_and_time** object. In **generic_network_model_representation** UoF: • change assertions of **generic_network_model** object. • change attributes and assertions of **submodel_usag**e object. • add elements in definition and EXPRESS specification of **network_node_relationshi**p.

Document Status Sheet for NRF-002-AP (Continued)

version, release, date	*reason for change*
	suppress **modelling_assumption**, **product_version_relationship**, **product_functional_decomposition_relationship** and **modelling_assumption_relationship** objects. In **bulk_results** UoF: • changed **run_context** attributes. • changed **aspect** definition and attributes. • added **aspect_characterization objec**t. • changed **generic_property_class** definition. • changed **descriptive_property_class** definition. • changed **aspect_scan** attribute. In **sensor_representation** UoF : • added **physical_sensor.** • added **virtual_sensor**. • added **calibration_law**. • added **sensor_limit.** In **sensor_oriented_results** UoF : • added **comparison_values**. • added **tabulated_values.** • added **column**.
v1, r1.3, 23-Aug-96	Significant restructuring of Clause 4, in line with the working session between Fokker Space and GOSET held 14-Aug-1996. Major modifications are: Application objects have been listed alphabetically. Moved discussion of NRF Results Space from Clause 4 to Clause 1.2 “Fundamental concepts and assumptions”. Removed EXPRESS specifications from Clause 4 descriptive specifications. Corrected “ordinate” to “abscissa”. Added **ato_campaign**, **ato_case** and **ato_phase** objects. Added security features Removed **sensor_oriented_results** UoF. The requirements in this UoF are covered by using the **scan_of_derived_values** and **property_class_for_functions**. These requirements also specified something like a tabular export format, which shall be a requirement for a postprocessor, not for the protocol. Moved the requirements of **aspect_characterization** object to the *roles* attribute of a **property_class** and to **scan_of_derived_values.** Moved requirements for **calibration_law** to **property_class_for_functions**, which provides a generic solution. Complete update of all EXPRESS-G diagrams in Annex G.
v1, r1.4, 15-Nov-96	• First release with mapping table and complete AIM (Clause 5 and Annexes A, B, C). • Clause 4 (ARM) was updated slightly to remove ambiguities and small errors. • **scan**: add INVERSE attribute of_aspect referring to an **aspect** • **aspect**: add assertion about the order of scans • **ato_phase**: add attribute of_case referring to an **ato_case** • **ato_case**: define attribute phases as an INVERSE attribute • **ato_case**: define unit_assignments as an INVERSE attribute • **ato_case**: changed attribute abscissa_class to refer to a **property_class_usage** • **ato_case**: add assertion to enforce abscissa_class.property to refer to a **property_class_scalar_quantitative** or a **property_class_scalar_descriptive**

Document Status Sheet for NRF-002-AP (Continued)

version, release, date	*reason for change*
	• **aspect**: remove the attribute of_case • **aspect**: remove the assertion • **list_of_descriptive_values**: replace the attribute of_case by the attribute applicable_for referring to a set of **property_class_usage** • **list_of_descriptive_values**: add an assertion to enforce the **list_of_descriptive_values** to be applicable for **property_class_scalar_descriptive** • **property_class_scalar_descriptive**: define attribute value_container as an DERIVE attribute referring to a SET[1:?] of **list_of_descriptive_values** • **list_of_functions**: replace the attribute of_case by the attribute applicable_for referring to a set of **property_class_usage** • **list_of_functions**: add an assertion to enforce the **list_of_functions** to be applicable for **property_class_scalar_functional** • **property_class_scalar_functional**: define attribute function_container as an DERIVE attribute referring to a SET[1:?] of **list_of_functions** • **property_class_tensor**: changed the attribute members into an INVERSE attribute referring to a SET[1:?] of **scalar_in_tensor** (the reason for change is that a property_class_scalar may be a member of one or more property_class_tensor, may be referred to by several members of a same property_class_tensor, and may have roles depending on each of its usages in the property_class_tensor) • **scalar_in_tensor**: add this new object to specify a member of a **property_class_tensor**, its position in the tensor and its meaning at this position • **property_class_tensor**: remove attribute tensor_mask (unuseful because of the addition of **scalar_in_tensor**) • **tensor_mask**: remove this entity (the information can be derived with considering the attribute position of the members of a **property_class_tensor**) • **tensor_mask_member**: remove this object • **tensor_mask_class**: remove this object. • **property_class_scalar**: remove attribute roles • **property_class_tensor**: remove attribute roles • **property_class_usage**: add this entity to characterize the usage of a **property_class** in an **ato_case** • **aspect**: change the attribute for_property_class to refer to a **property_class_usage** • **model_component_sequence**: define the attribute root_model as a DERIVE attribute • **parameterized_function**: restrict the attributes parameter_properties and result_property to the type **property_class_scalar_quantitative** • **model_represents_product_relationship**: add **network_model** in the list of types allowed for the attribute model_constituent in order to have a definition consistent with the assertion **network_model_requires_product_definition** • **approvals_are_assigned**: add this assertion for consistency with ISO 10303-41 (definition of approval) and with ISO 10303-203 Furthermore, the *Space-domain Integrated Resources* (SIR) were developed as a separate document (SIR-001-IR, v1, r1.0). The SIR define generic resources which can be used by all Application Protocols in the space domain.

Changes with respect to the previous release are indicated by vertical bars in the margin.

Document Change Record

DCR number		
date		
originator		
approved by		
document title		STEP-NRF – Network-model Results Format
document reference id		NRF-002-AP
document version/release		Version 1 Release 1.4
page	*section*	*reason for change*

Table of Contents

Foreword

This document is a STEP based application protocol for the electronic exchange of bulk results data associated with network models. A network model in this context is a generic representation of an engineering object (or a collection of related engineering objects) by a collection of discrete network nodes and relationships between these nodes. The bulk results are characteristic, predicted, assigned or observed properties for components of the engineering object(s), which are sampled during an analysis, test or operation run.

The following annexes are normative: A, B, C, D, E.

The following annexes are informative: F, G, H, J, K, L, M.

This Application Protocol follows to a large extent the guidelines and recommendations of [STEP-AP-Guide] “Guidelines for the Development and Approval of STEP Application Protocols” and [STEP-Sup-Dir] “Supplementary directives for the drafting and presentation of ISO 10303”.

Clauses 3 and 4 were produced using many of the concepts and approaches documented in [ESABASE-Lib], [SET-ATS], [GFF], [TDH], [TFCC], [ESATAN-UM], [CDF], [netCDF], [HDF], and in particular [GEM-Core].

Annex M provides the details on all these references.

Introduction

0.1 Brief introduction on the STEP standard

STEP is the casual name for the ISO 10303 standard and stands for Standard for the Exchange of Product model data.Work on the standard started in 1984 and is still in progress with participants from all over the world. This introduction is compiled with elements from [STEP-1], [Kuiper-94] and [Fowler-95].

0.1.1 The objectives of STEP

STEP part 1 says: “ISO 10303 is a series of International Standards for the computer-interpretable representation and exchange of product data. The objective is to provide a neutral mechanism capable of describing product data throughout the life cycle of a product, independent from any particular system. The nature of this description makes it suitable not only for neutral file exchange, but also a basis for implementing and sharing product databases and archiving.” [STEP-1].

In other words the STEP methodology has been developed to meet industry requirements for standard data specifications that support:

- long term storage and retention of product information;
- reduction and elimination of “islands of automation”;
- independence of data from the software tools which create or consume information;
- communication of product information between departments, disciplines, and enterprises.

In addition, the fact that STEP is a standard introduces additional requirements, in that the specification developed to fulfil these requirements should be stable, extensible, and publicly available.

0.1.2 Principles

The STEP methodology is based on a small number of fundamental principles:

- STEP defines an architecture for product data, providing stability and extensibility;
- STEP supports and requires traceability of data to industry needs;
- The role of STEP is the standardisation of industry application semantics;
- STEP defines the requirements for implementation of product data exchange, based on a separation of data specifications (the logical definition) from implementation forms (the physical realisation);
- STEP defines the requirements for the assessment of conformance of implementations.

0.1.3 The STEP architecture

The architecture of STEP results from the principles stated above. The complete architecture of STEP is described in the *ISO 10303 Architecture and Reference Manual*, which is currently being prepared for publication. Below the main elements of the architecture will be highlighted.

Layered approach with functional series of standard documents

The STEP standard takes a layered approach to product data modelling. The different layers are called series and these contain parts which are related by the function they perform. The series are given in Table 1.

Each series adds specific aspects to the product data model description. The higher numbered layers build further on concepts defined in the lower numbered layers. The EXPRESS language is used to formally define the data models (*schemas* in STEP parlance) of the parts in layers 4, 5, 6 and 7. Application protocols are directed for use by a specific user community, engineering discipline or branch of industry. The standard also includes a formal methodology for conformance testing of STEP-based products that claim to implement one of the application protocols.

Table 1 - The series of STEP standard documents

layer	*series title and part number range*	*example standard documents*
1	Description Methods *10 series*	ISO 10303 Part 11 The EXPRESS Language Reference Manual
2	Implementation Methods *20 series*	ISO 10303 Part21 Clear Text Encoding of the Exchange Structure ISO 10303 Part22 Standard Data Access Interface
3	Conformance Testing Methodology and Framework *30 series*	ISO 10303 Part31 General Concepts
4	Integrated Generic Resources *40 series*	ISO 10303 Part41 Fundamentals of Product Description and Support ISO 10303 Part42 Geometric and Topological Representation
5	Integrated Application Resources *100 series*	ISO 10303 Part105 Kinematics
6	Application Interpreted Constructs *500 series*	ISO 10303 Part509 Manifold Surface Shape Representation
7	Application Protocols *200 series*	ISO 10303 AP 203 Configuration Controlled Design
8	Abstract Test Suites *300 series*	ISO 10303 Part 303 Abstract Test Suite – Configuration Controlled Design (for AP 203)

Key elements of the STEP architecture

Figure 0–1 provides a high level summary of the key elements of the STEP architecture. The arrows in Figure 0–1 specify the *existence dependence* between the elements, i.e. an element at an arrow's tail is dependent on an element at its head.

After the figure a short description of each of the key elements is given.

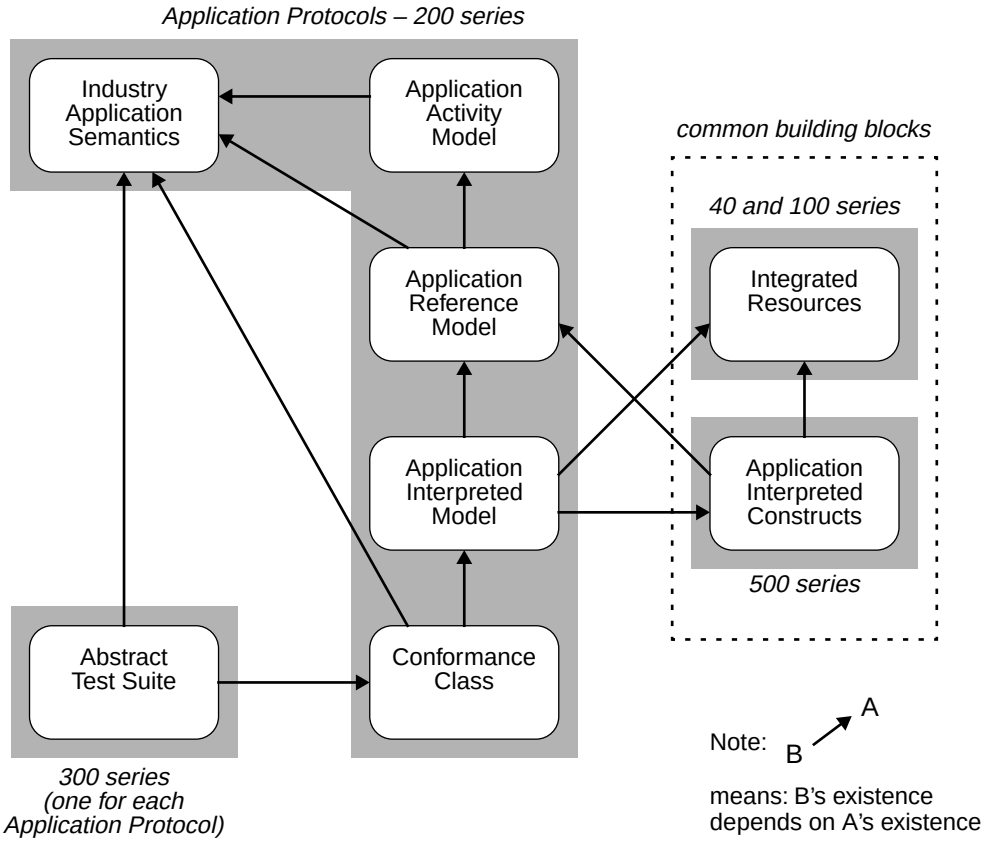

Figure 0–1 - Key elements of the STEP architecture – the grey areas denote the relation between these elements and the STEP documentation structure

Industry Application Semantics

Figure 0–1 shows that the key element upon which all other elements of the STEP architecture depend is industry application semantics. It identifies the processes and data that are essential to an industrial application domain – in other words the industrial needs and semantics that are particular to data exchange in some industry application domain. It must be possible to trace back the concepts and definitions used in the other architectural elements to the needs and semantics in the Industry Application Semantics.

Application Activity Model

Industry application semantics are defined more formally by reference to an Application Activity Model (AAM). This model is created with the IDEF0 activity modelling technique and supports the analysis of the activities and information flows within the industry application. The AAM provides a context model for the application domain and the activities and information flows that are "in scope" or "out of scope" are explicitly declared. The "in scope" information flows can then be elaborated in further detailed STEP analysis and specification models – in particular the Application Reference Model (ARM). It should be noted that the role of the AAM is to capture the activities within an industry application ("what is done"), not the detailed processes ("how it is done"). The latter are likely to vary between organisations, or with time as the result of continuous improvement or business process re-engineering.

Application Reference Model

The second element of the STEP architecture that results from detailed analysis of the product data exchange requirements in the industrial application domain, is the Application Reference Model (ARM). This is a detailed specification of the data objects (entities and attributes) and their relationships, that completely define the "in scope" information flows, which were identified in the AAM. This specification is prepared through analysis of requirements identified by experts in the

industry application – sometimes referred to as “domain experts”. These requirements are therefore described using the terminology of the application domain. They form the basis for further development as well as for review and validation, e.g. by peer experts.

Application Interpreted Model

The data exchange requirements laid down in the ARM are translated – by a “STEP expert” – into a normative STEP data specification in the Application Interpreted Model, as much as possible through selection and constraint of the common building blocks: the Integrated Resources and the Application Interpreted Constructs. This re-use of standard data constructs across a wide range of industry requirements results in a high degree of consistency and integration across models, and enables potential re-use of the software code used in interfaces and the potential sharing of common data across application domains. The AIM is defined using the EXPRESS language, which is computer interpretable and therefore enables file-based exchange according to the STEP Part 21 physical file format and/or data access using the STEP Part 22 SDAI.

Integrated Resources

The Integrated Resources are specified in context independent models and form standard data constructs that are used in the creation of an AIM. They are data models that reflect and support the common requirements of many different product data application areas. Together the Integrated Resources constitute a single, logical, conceptual product data model. They are, however, not themselves intended for direct implementation: they define re-usable building blocks that are intended to be combined and refined to meet specific requirements stated in an ARM. Integrated Resources are specified in EXPRESS.

Application Interpreted Constructs

The Application Interpreted Constructs are a relatively late addition to the STEP architecture. They capture the fulfilment of a collection of requirements that are common to two or more ARMs, i.e. shared by two or more industry application domains. Application Interpreted Constructs are also specified in EXPRESS and explicitly identify the potential for shared data or so-called *interoperability* between two or more Application Protocols / industry application domains.

The difference between Integrated Resources and Application Interpreted Constructs can explained as follows:

- Integrated Resources were developed – and are still being developed and refined – on the basis of an a priori conception by product data modelling experts of what common product data modelling resources would be needed across a wide range of application domains. As stated before Integrated Resources are not intended for direct implementation in an AIM, but often need to be combined and refined first.
- Application Interpreted Constructs are the common collections of requirements in the ARM and their corresponding implementation in the AIM that are “discovered” during the development of various Application Protocols. Application Interpreted Constructs are a ready-for-implementation kind of “plug-in” modules that can be included in an AIM.

Conformance Classes

An Application Interpreted Model gives the normative specification for data to be exchanged between computer applications. This provides the scope and boundaries for implementations of product data exchange that conform to STEP, and also the cope and boundaries for testing implementations. In order to meet the needs of differing computer systems used within a given industrial application, whilst maintaining consistency of implementation and testing, two or more Conformance Classes may be specified for an AIM. A Conformance Class defines a subset of the AIM that may be used as the basis for implementation and testing. These subsets define the minimum conforming implementation based on the AIM; implementations based on any other subsets are not considered to be conforming.

Abstract Test Suites

The importance of testing and testability within STEP is reflected by a standardised framework and methodology for conformance testing. The Abstract Test Suite is the manifestation of the needs of testing within the STEP architecture. An Abstract Test Suite specifies, in non-specific or parameterised form, the test cases that will be used in assessing the conformance of an implementation to the data

specification contained in an AIM and thee other elements of the STEP architecture upon which an AIM depends. Experience in other domains, such as the OSI standard for Open Systems, has shown that standardisation of Abstract Test Suites is an essential prerequisite to repeatability and consistency of testing, and therefore of mutual recognition of test results across regional or national bodies.

0.1.4 The basis of the STEP product data model

The consistency of data specifications within STEP for a wide range of industry applications – Application Protocols – is ensured by the reuse of common Integrated Resources. The Integrated Resources themselves are based in a formalised framework for product data, sometimes referred to as the Generic Product Data Model. The framework defines the basis of all data specification that are standardised within STEP.

Elements of data specifications (or "constructs") are taken to be the representation of facts about objects in the real world. The Generic Product Data Model framework is based on a classification of the types of data that describe products. The classification identifies five major types of data:

- *Application context* – data that defines the purpose for which product information is created, and the types of product, disciplines, and life-cycle stages for which such information is valid. The use of an application context allows e.g. data that represents an "as designed" product to be distinguished from that for an "as built" configuration, etc.
- *Product definition* – data that identifies products, including variants, categories and/or life-cycle "views" of products. Product definition data also includes that which relates to the structure of products, in terms of assembly structures, configurations, effectivities, bill of material, etc.
- *Product property definition* – data that characterises products by their properties, independent of the representation of such properties. For example, it is possible to identify the shape of an object, or aspects of the shape, as a property of the object, without providing a detailed description of the shape using a CAD model or an engineering drawing. Examples of product properties are: shape, surface finish, constituting material.
- *Product property representation* – data that represents the properties of a product, including multiple representations of the same property. For example, the shape of an object may be identified, and then described in many different ways: a 3D CAD model, a physical mock-up, an engineering drawing, a technical illustration and a mathematical model are different representations of the same shape.
- *Product property presentation* – data that defines the presentation of product information to support human communication. For example, the shape of an object (which is a product property) is represented by coordinate values, curves, surfaces, etc.; then this representation is presented by assigning colours, line fonts and a viewpoint, and displaying the resulting picture on a workstation.

This classification of product data is the basis for all STEP data specifications. It is the framework upon which all the Integrated Resources are built, and is reflected in the Application Interpreted Models (AIMs) of all Application Protocols. The models that capture this framework embody the principle of existence dependence, which ensures that all product information is related to an identified product and ultimately to an application context. This classification structure is summarised graphically in Figure 0–2.

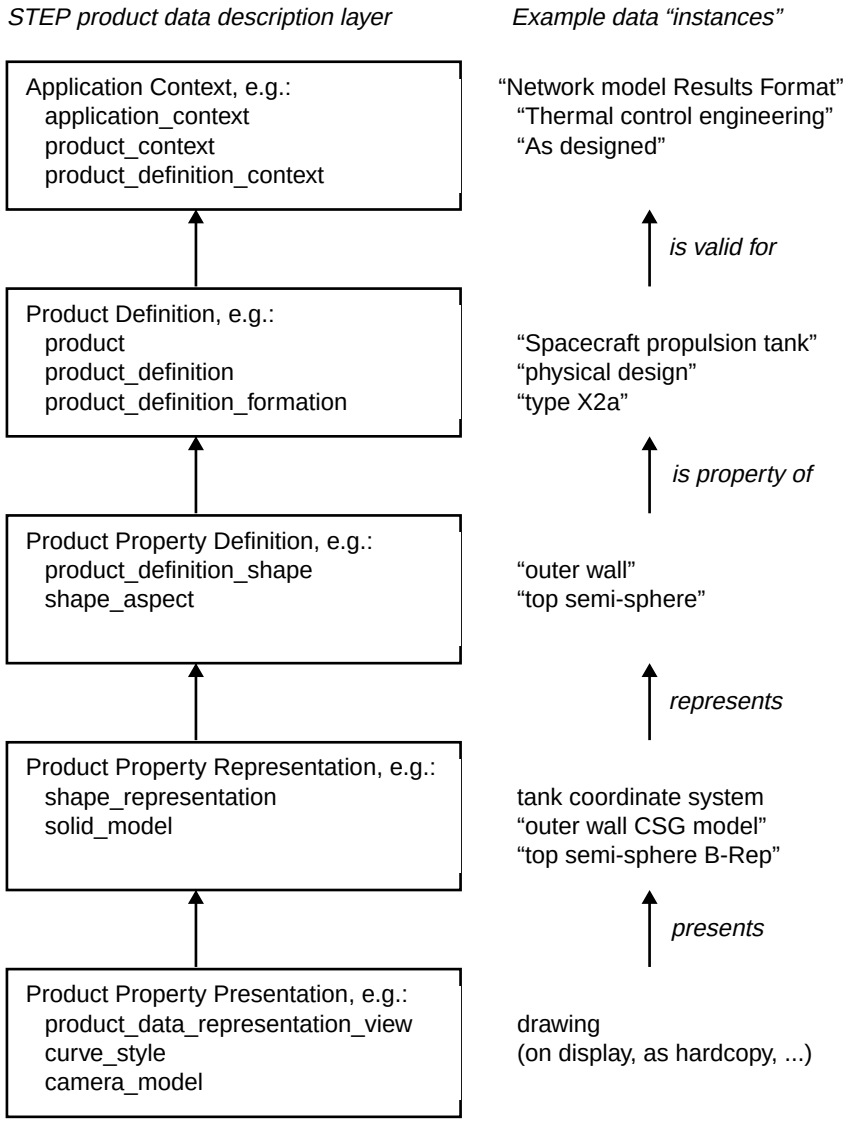

Figure 0–2 - Product data type classification and existence dependence in STEP data models

This principle can lead to models that are at first sight counter-intuitive: rather than stating that a product has a shape, a shape is "of" a product. However, simple analysis of this example shows that the existence dependent form of the model requires that a shape is always the shape of a product. Similarly, at a lower level in the structure, STEP does not allow the existence of "geometry" data just as collections of points, lines, curves, etc. Through the existence dependent structures in the STEP models, such a collection of geometry data must be the representation of some property, that is related to the definition of a product, that has validity in some application context. Thus the basic structure of the STEP models satisfies and enforces the principle identified above: that all product data should be traceable to an industrial need.

0.1.5 Realisation of a STEP solution for product data exchange

One of the main features of the STEP standard is the division between *logical* and *physical* representation of model data.This helps in achieving a clean development process which is best illustrated by the following diagram.

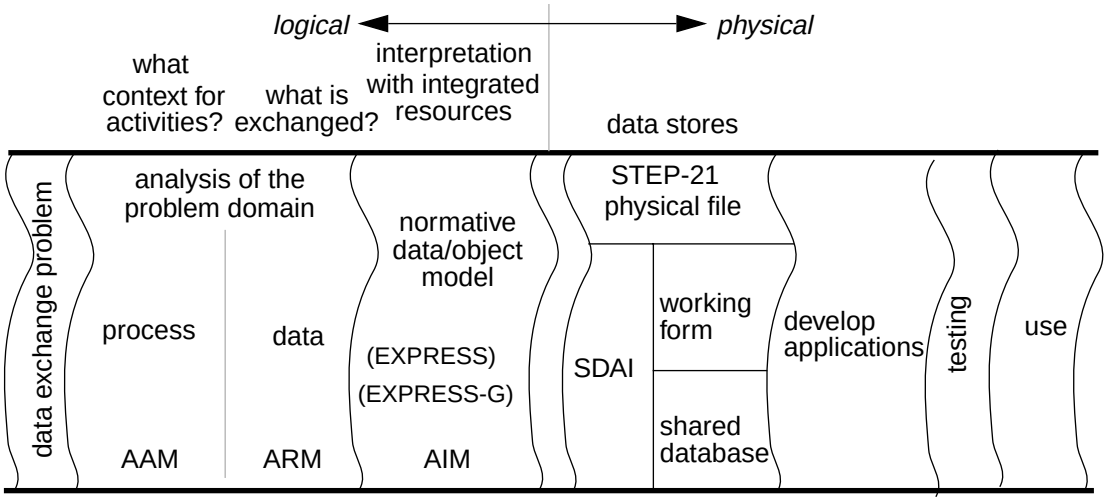

Figure 0–3 - Illustration of the development steps needed to reach a working STEP product data exchange solution and different elements of the STEP standardisation methodology (adapted from a diagram by P. Kuiper – TNO)

Actual data exchange takes place through the physical implementation in either a [STEP-21] physical file, a so-called working form or shared database. The physical implementation is derived in a standardised way from the logical definition in an Application Interpreted Model (AIM) of a certain Application Protocol and can possibly be accessed through an SDAI [STEP-22] compliant programming interface, for which a significant part could have been computer-generated from the AIM EXPRESS schema by a suitable STEP toolkit.

0.1.6 Status of the STEP standard and its implementations

The STEP standard is becoming mature and the First Release (Parts 1, 11, 21, 31, 41, 42, 43, 44, 46, 101, 201, 203) has been issued per September 1994 as International Standard (IS). Commercial development tools are available on the market and most major CAx software vendors are now offering STEP-based import/export facilities or have announced to support the STEP standard.

Clause 1 - Scope

1.1 Definition of Scope

This application protocol specifies the STEP resources necessary for the electronic exchange of bulk results data associated with network models. A network model in this context is a generic representation of an engineering object (or a collection of related engineering objects) by a collection of discrete network nodes and relationships between these nodes. The bulk results are characteristic, predicted, assigned or observed properties for components of the engineering object(s), which are sampled during an analysis, test or operation run.

This application protocol is aimed at batch exchange of bulk results data, after completion of a run. The design does not include special features for handling of streams of result data during the execution of a run.

The following are within the scope of this application protocol:

- The representation of an engineering object by a network model of discrete nodes and relationships between those nodes.
- A hierarchical tree structure of network models and submodels.
- The definition and representation of properties of engineering objects. Both quantitative properties (with numerical value) and descriptive properties (with descriptive content) are supported.
- The representation of values of properties as scalars, vectors and tensors.
- The definition and representation of analysis, test and operation runs which produce bulk results.
- The definition and representation of product structure, in the form of assembly trees, and the relationships between items in the product structure and in the network model representation.

The following are outside the scope of this application protocol:

- Discipline dependent specialisations of network models, such as: thermal lumped parameter models, structural finite element models, computational flow dynamics models.

1.2 Fundamental concepts and assumptions

1.2.1 The stages of activity relevant to analyses, tests and operations

When performing analyses, tests or operations on a product of interest, three major stages can be distinguished:

1) The *definition* of the *mission* and the flowdown into a set of established *requirements* for the product – i.e. the definition of desired or needed behaviour.
2) The *definition* of analysis, test or operation *cases*, which define a scenario, initial and boundary conditions and a representation model of the product of interest. Such scenarios, conditions and models mostly consist of idealisations and/or extreme combinations of conditions. These are used for the following purposes:
   - In analysis, to predict how the product will operate.
   - In analysis or test, to verify that the product operates or will operate conform its requirements.
   - In actual operation, to measure how the product is performing while deployed in reality.

   An analysis, test or operation *campaign* may collect a number of related *cases*.
   A *case* may be decomposed into a number of analysis, test or operation *phases*.
3) The *execution* of analysis, test or operation *runs*, in which the bulk results data are produced.

   A *run sequence* may collect a sequence of related *runs*.

In this protocol stages 2) and 3) are in scope, but stage 1) is not in scope.

The table below summarises the three stages and the most important concepts that play a role in each stage.

Stage 1 – Definition of mission and requirements	*Stage 2 – Definition of analyses, tests and operations*	*Stage 3 – Execution of analyses, tests and operations*
mission	campaign	
requirement	case	run sequence
	phase	run, scan
product structure	product structure	
	network model	
	property class	property value
	scalar, vector, tensor	
	parameterised function	

1.2.2 The NRF results space

The conceptual architecture for storing results is very simple: results are stored as characteristic, predicted, assigned or observed properties of model components in a results space. In this results space, a property is identified by coordinates in only three dimensions:

- *what* is the property class for which values are stored in the dataset?
- *where* in the model – that is: for which model components – does this property apply ?
- *when* – that is: at which scan abscissa along the progression of the run – does a property class for a particular model component have a particular value?

The results space can be visualised as in the diagram of Figure 1–1.

In the real world a number of factors complicate this simple concept:

- In order to interpret the results data correctly, an exchange dataset needs to provide an application and its user with sufficient (meta-)information *about* the exchange dataset and the properties.
- The results are not restricted to just bare numbers, so it is necessary to include information about *how* the results are represented. Even when the values of properties can be expressed as numeric scalars, it is necessary to include information about what these numbers signify. In the case of physical quantities one needs to know in which unit the values are expressed. Besides scalars, the values can also be vectors, tensors or descriptive data, for which representation structures have to be specified.

A final consideration in the design of this protocol is that of efficiency, which is very important for bulk results data where models have many thousands of components, and a single component's property value may be sampled thousands of times in a run. The straightforward approach of storing a potential property value for every property class, for every model component, at every possible scan abscissa would lead to very inefficient storage because:

- Property classes do not necessarily apply to all model components.
- The observation of property values is not necessarily synchronized as the values of properties are not always obtained at the same scan abscissa. There is a good chance that it is impossible to determine the full state – which contains all applicable property values for all model components – at a particular scan abscissa.

Because of these reasons, valid and/or defined property values may be distributed very sparsely throughout the results space.

The NRF protocol contains special features to achieve efficient storage, even for sparsely distributed data:

- the use of **aspect**s prevent storage of data for meaningless combinations of model components, properties and scans

- use of masks allow indication of the presence of data, preventing the storage of indeterminate values for undefined data
- referencing common data structures (like text strings) prevents data duplication

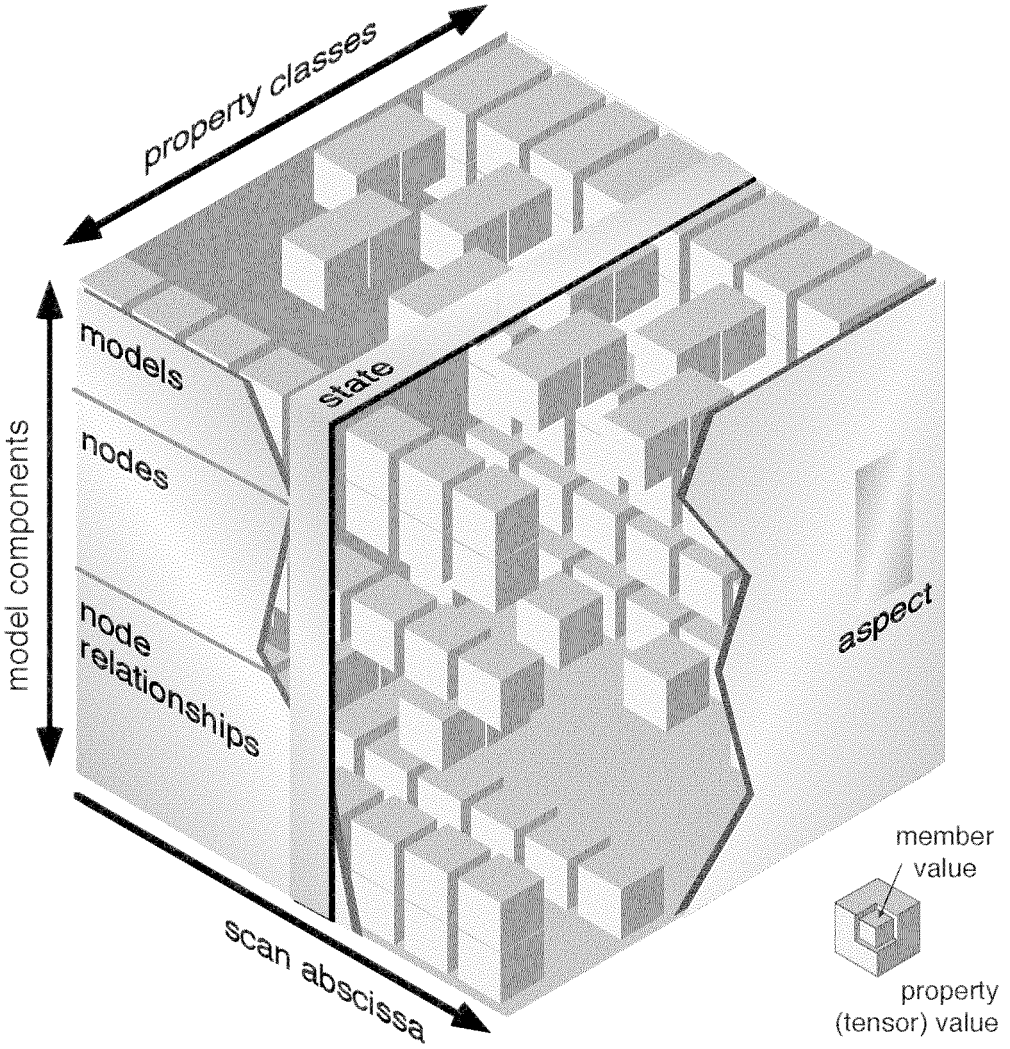

Figure 1–1 - Schematic illustration of the bulk results space and its three dimensions: model components, property classes and scan abscissa

Clause 2 - Normative References

The following documents, ISO standards or working drafts contain provisions which, through reference in this text, constitute provisions of this Application Protocol. At the time of development, the editions indicated between parenthesis were valid.

[STEP-1]	ISO 10303-1:1994 (IS) Overview and fundamental principles
[STEP-11]	ISO 10303-11:1994 (IS) Description methods: The EXPRESS language reference manual
[STEP-21]	ISO 10303-21:1994 (IS) Clear text encoding of the exchange structure Note: Also known as “STEP Physical File”.
[STEP-31]	ISO 10303-31:1994 (IS) Conformance testing methodology and framework
[STEP-41]	ISO 10303-41:1994 (IS) Integrated generic resources: Fundamentals of product description and support
[STEP-42]	ISO 10303-42:1994 (IS) Integrated generic resources: Geometric and topological representation
[STEP-43]	ISO 10303-43:1994 (IS) Integrated generic resources: Representation structures
[STEP-44]	ISO -10303-44:1994 (IS) Integrated generic resources: Product structure configuration
[STEP-45]	ISO/DIS 10303-45 (DIS) Integrated generic resources: Materials
[STEP-46]	ISO 10303-46:1994 (IS) Integrated generic resources: Visual presentation
[STEP-104]	ISO 10303-104:1996 (CD) Integrated application resources: Finite element analysis

Note: the edition (release status) of the documents is given between parentheses, using the following abbreviations:

WD	Working Draft
CD	Committee Draft
CDB	Committee Draft, in Ballot
CDBE	Committee Draft, Ballot Ended
CDC	Committee Draft, for Comment
CDCE	Committee Draft, Comments Ended
DIS	Draft International Standard
IS	International Standard

Clause 3 - Definitions and Abbreviations

3.1 Terms Defined in ISO 10303-1

3.1.1 **abstract test suite** – a part of this International Standard that contains the set of abstract test cases necessary for conformance testing of an implementation of an application protocol.

3.1.2 **application** – a group of one or more processes creating or using product data.

3.1.3 **application activity model (AAM)** – a model that describes an application in terms of its processes and information flows.

3.1.4 **application context** – the environment in which the integrated resources are interpreted to support the use of product data in a specific application.

3.1.5 **application interpreted model (AIM)** – an information model that uses the integrated resources necessary to satisfy the information requirements and constraints of an application reference model, within an application protocol.

3.1.6 **application object** – an atomic element of an application reference model that defines a unique concept and contains attributes specifying the data elements of the object.

3.1.7 **application protocol** – *a document complying with [AP-Guide]* that specifies an application interpreted model satisfying the scope and information requirements for a specific application.

3.1.8 **application reference model (ARM)** – an information model that describes the information requirements and constraints of a specific application.

3.1.9 **application resource** – an integrated resource whose contents are related to a group of application contexts.

3.1.10 **assembly** – a product that is decomposable into a set of components or other assemblies from the perspective of a specific application.

3.1.11 **component** – a product that is not subject to decomposition from the perspective of a specific application.

3.1.12 **conformance class** – a subset of an application protocol for which conformance may be claimed.

3.1.13 **conformance requirement** – a precise, text definition of a characteristic required to be present in a conforming implementation.

3.1.14 **data** – a representation of information in a formal manner suitable for communication, interpretation, or processing by human beings or computers.

3.1.15 **data exchange** – the storing, accessing, transferring, and archiving of data.

3.1.16 **data specification language** – a set of rules for defining data and their relationships suitable for communication, interpretation, or processing by computers.

3.1.17 **exchange structure** – a computer-interpretable format used for storing, accessing, transferring, and archiving data.

3.1.18 **generic resource** – an integrated resource whose contents are context-independent.

3.1.19 **implementation method** – a part of this International Standard that specifies a technique used by computer systems to exchange product data that is described using the EXPRESS data specification language [STEP-11].

3.1.20 **information** – facts, concepts, or instructions.

3.1.21 **information model** – a formal model of a bounded set of facts, concepts or instructions to meet a specified requirement.

3.1.22 **integrated resource** – a part of this International Standard (i.e. ISO 10303) that defines a group of resource constructs used as the basis for product data.

3.1.23 **interpretation** – the process of adapting a resource contruct from the integrated resources to satisfy a requirement of an application protocol. This may involve the addition of restrictions on attributes, the addition of constraints, the addition of relationships among resource constructs and application constructs, or all of the above.

3.1.24 **PICS proforma** – a standardised document in the form of a questionnaire, which, when completed for a particular implementation, becomes the protocol implementation conformance statement.

3.1.25 **presentation** – a recognizable visual representation of product data.

3.1.26 **product** – a thing or substance produced by a natural or artificial process.

3.1.27 **product data** – a representation of information about a product in a formal manner suitable for communication, interpretation, or processing by human beings or by computers.

3.1.28 **product information** – facts, concepts, or instructions about a product.

3.1.29 **product information model** – an information model which provides an abstract description of facts, concepts and instructions about a product.

3.1.30 **protocol implementation conformance statement (PICS)** – a statement of which capabilities and options are supported within an implementation of a given standard. This statement is produced by completing a PICS proforma.

3.1.31 **resource construct** – a collection of EXPRESS language entities, types, functions, rules and references that together define a valid description of an aspect of product data.
Refer to ISO 10303 part 11 for the reference manual of the EXPRESS language.

3.1.32 **structure** – a set of interrelated parts of any complex thing, and the relationships between them.

3.1.33 **unit of functionality** – a collection of application objects and their relationships that defines one of more concepts within the application context such that removal of any component would render the concept incomplete or ambiguous.

3.2 Terms Defined in ISO 10303-31

3.2.1 **postprocessor** – a software unit that translates product information from an independent public domain product data format to the internal format of a particular computer system.

3.2.2 **preprocessor** – a software unit that translates product information from the internal format of a particular computer system to an independent public domain product data format.

3.3 Terms Defined in ISO 10303-45

3.3.1 **material** – the substance or substances of which product is composed or made.

3.3.2 **material property** – a product characteristic which depends upon the material or materials comprising the product.

3.3.3 **material designation** – an identifier which is assigned by agreement.

3.4 Terms Defined in this AP

3.4.1 **class** – a named category of items which share common characteristics and behaviour.

3.4.2 **exchange dataset**– coherent and valid set of instances of entities conforming to the AIM schema defined in this AP. The dataset is instantiated in the form of a [STEP-21] physical file, a working form or a shared database. An exchange dataset can be accessed through an SDAI programming interface, as defined in [STEP-22].

3.4.3 **model** – a representation in software or hardware of one or more properties of a product and possibly its operational environment for the purpose of design, analysis or verification. By its nature a model is an idealised representation of the product.

3.4.4 **property** - a quality or attribute belonging and especially peculiar to an object, common to all members of a class.

3.4.5 **qualifier** - a word or word group that limits or modifies the meaning of another word or word group [Webster].

3.4.6 **root-model** – The highest level model in a hierarchical model/submodel tree. The root-model comprises directly or indirectly all submodels.

3.4.7 **run** – an elementary analysis, test or operation step with a clearly defined start and end, performed in the framework of an analysis study, a test or an operation. In a run an analysis model, a test article or an operating product is analysed, tested or operated for a particular purpose and in a particular environment, using a particular analysis tool, a test facility or (when applicable) an operational facility. A run produces analysis, test or operation results.

3.4.8 **sensor** – a means or device for observation of a property.

3.4.9 **state** - mode or condition of being [Webster].

3.4.10 **submodel** – a next lower level model in a hierarchical model/submodel tree. A submodel may contain other (yet lower level) submodels. The model/submodel tree forms an acyclic graph.

3.4.11 **submodel occurrence** – a reference to a model at a level lower than root-model level in a hierarchical model/submodel tree. The submodel occurrence has some attributes in addition to the model it references: e.g. its usage name and – for geometric models – its location and orientation with respect to its supermodel.

3.4.12 **supermodel** – a next higher level model in a hierarchical model/submodel tree. A supermodel contains one or more submodels. The model/submodel tree forms an acyclic graph.

3.4.13 **tensor** - a generalization of the concept of vector that consists of a set of components usually having multiple rows of indices that are functions of the coordinate system and have invariant properties under transformation of the coordinate system. [Webster]

3.4.14 **unit** - a determinate quantity (as of length, time, heat, value, or housing) adopted as a standard of measurement [Webster]

3.5 Abbreviations

AAM	Application Activity Model
AIM	Application Interpreted Model
AP	Application Protocol
ARM	Application Reference Model
B-rep	Boundary Representation Solid Model
CNES	Centre National d'Etudes Spatiales – French Space Agency
ESA	European Space Agency
ESTEC	ESA - European Space Technology Centre
ISO	International Organization for Standardization
OO	Object Oriented
SDAI	Standard Data Access Interface (defined in [STEP-22])
SET	*Standard d'Echange et de Transfert* (French Standard AFNOR Z68-300)
SET-ATS	SET Protocol d'Application Thermique Spatiale
SI	*Système International des Unités* - International System of Units
STEP	Standard for the Exchange of Product Model Data (ISO 10303)
STEP-TAS	STEP-based Thermal Analysis for Space
TBD	To be defined
URD	User Requirements Document
UoF	Unit of Functionality

Clause 4 - Information Requirements

This clause specifies in detail what kind of data needs to be exchanged in the domain of results for network models. The information requirements are grouped in three subclauses: Units of Functionality, Application Objects and Application Assertions.

A graphical representation of the information requirements is given in Annex G “Application Reference Model (Informative)” on page G-1.

4.1 Units of Functionality

This subclause specifies the units of functionality for the Network Model Results Format AP. This protocol specifies the following units of functionality:

1) **product_structure** UoF
2) **network_model_representation** UoF
3) **bulk_results** UoF
4) **parameterized_functions** UoF
5) **physical_quantities_and_units** UoF
6) **date_and_time** UoF
7) **general_support** UoF

The units of functionality and a description of the functions that each UoF supports are given below. The application objects included in the UoFs are defined in subclause 4.2.

4.1.1 product_structure UoF

The **product_structure** UoF captures the structure of the product being modelled – e.g. the product assembly tree or a functional decomposition tree. The definition of this UoF already follows closely the product structure definition in [STEP-41] and [STEP-44].

NOTE 1 – This Unit of Functionality is defined following as closely as possible the concepts for product and product structure definition in [STEP-41] and [STEP-44].

WORKING NOTE 1 – This UoF is intended to be shared among the STEP-TAS and NRF protocols.

The following application objects are used by the **product_structure** UoF:

- **product**
- **product_context**
- **product_definition**
- **product_definition_context**
- **product_next_assembly_usage_relationship**
- **product_version**

4.1.2 network_model_representation UoF

The **network_model_representation** UoF collects the definitions needed for a generic representation of engineering objects as a hierarchical structure of models containing a network of discrete nodes and relationships between these nodes.

WORKING NOTE 2 – This UoF is intended to be shared among the STEP-TAS and NRF protocols.

The following application objects are used by the **network_model_representation** UoF:

- **model_represents_product_relationship**
- **modelling_assumption** (not used in NRF)
- **modelling_assumption_relationship** (not used in NRF)
- **network_model**
- **network_model_component**
- **network_model_constituent**
- **network_node**

- **network_node_class**
- **network_node_relationship**
- **network_node_relationship_class**
- **network_node_relationship_usage**
- **network_node_usage**
- **submodel_usage**

4.1.3 bulk_results UoF

The **bulk_results** UoF collects the application objects needed for the identification and definition of analysis, test or operation campaigns, cases, phases and runs and the results data they produce. The results data are characteristic, predicted, assigned, observed and/or derived properties of network models, of which the values may need to be represented as scalars, vectors or higher order tensors.

The following application objects are used by the **bulk_results** UoF:

- **abscissa_sequencing_type**
- **ato_aspect**
- **ato_campaign**
- **ato_case**
- **ato_phase**
- **lifetime_type**
- **list_of_descriptive_values**
- **list_of_functions**
- **model_component_sequence**
- **property_class**
- **property_class_scalar**
- **property_class_scalar_descriptive**
- **property_class_scalar_functional**
- **property_class_scalar_quantitative**
- **property_class_tensor**
- **property_class_usage**
- **property_name**
- **property_role**
- **property_symmetry**
- **run**
- **run_sequence**
- **scalar_in_tensor**
- **scalar_value**
- **scan**
- **scan_derivation_procedure**
- **scan_mask**
- **scan_of_derived_values**
- **scan_of_sampled_values**
- **tensor_mask**
- **tensor_mask_member**
- **tensor_mask_name**
- **unit_assignment**

4.1.4 parameterized_functions UoF

The **parameterized_functions** UoF contains the application objects needed to store functions of one or more independent parameters, which are physical quantities. Functions which can be expressed in tabular or polynomial form or as textual formulae are considered.

WORKING NOTE 3 – This UoF is intended to be shared among the STEP-TAS and NRF protocols.

WORKING NOTE 4 – This UoF is included for extension of the NRF protocols with test design and definition data.

The following application objects are used by the **parameterized_functions** UoF:

- **cyclic_tabular_function**
- **parameterized_function**
- **parameterized_function_with_formula**
- **polynomial_function**
- **tabular_function**
- **tabular_interpolation_type**

4.1.5 physical_quantities_and_units UoF

The **physical_quantities_and_units** UoF captures the definition and representation of physical quantities, their dimensions, units and values.

NOTE 2 – The objects defined in this UoF correspond one-to-one with those defined in the **measure_schema** of [STEP-41]. Here only copies of the directly used objects are given.

The following application objects are used by the **physical_quantities_and_units** UoF:

- All objects from the **measure_schema** defined in clause 4.14 of [STEP-41].

In particular:

- **unit**

4.1.6 date_and_time UoF

The **date_and_time** UoF allows representation combinations of calendar date and time of day to be defined.

NOTE 3 – The objects defined in this UoF correspond one-to-one with those defined in the **date_time_schema** of [STEP-41]. Here only copies of the directly used objects are given.

NOTE 4 – The [STEP-41] **ordinal_date**, **day_of_week** and **week_of_year** objects are out of scope.

The following application objects are used by the **date_and_time** UoF:

- **calendar_date**
- **coordinated_universal_time_offset**
- **date**
- **date_and_time**
- **local_time**

The objects in this UoF are taken as much as possible literally from [STEP-41].

4.1.7 general_support UoF

The **general_support** UoF contains application objects that are used to support the use of the principal application objects listed in the other Units of Functionality in this subclause.

The following application objects are used by the **general_support** UoF:

- **address**
- **approval**
- **document**
- **document_type**
- **document_usage_constraint**
- **organization**
- **organizational_address**
- **organizational_project**
- **person**
- **person_and_organization**
- **personal_address**
- **security_classification_level**

The objects in this UoF are taken as much as possible literally from [STEP-41] and [STEP-203].

4.2 Application objects

4.2.1 abscissa_sequencing_type

An **abscissa_sequencing_type** is an enumeration that indicates how the **abscissa_value** of **scan**s progresses over the list of **scan**s within an **ato_aspect**. It indicates one of the following:

- `strictly_decreasing`, indicating that the **abscissa_value**s are strictly decreasing, meaning that the **abscissa_value** of a **scan** is always smaller than the previous one from the **scan**s list of an **ato_aspect**.
- `monotonic_decreasing`, indicating that the **abscissa_value**s are monotonic decreasing, meaning that the **abscissa_value** of a **scan** is always smaller than or equal to the previous one from the **scan**s list of an **ato_aspect**.
- `monotonic_increasing`, indicating that the **abscissa_value**s are monotonic increasing, meaning that the **abscissa_value** of a **scan** is always larger than or equal to the previous one from the **scan**s list of an **ato_aspect**.
- `strictly_increasing`, indicating that the **abscissa_value**s are strictly increasing, meaning that the **abscissa_value** of a **scan** is always larger than the previous one from the **scan**s list of an **ato_aspect**.

4.2.2 address

An **address** is the place where people and organizations are located and/or where they can be contacted. An **address** may be associated with **person**s or **organization**s through **personal_address** or **organizational_address**.

Attributes:

- *internal_location*: OPTIONAL label string, specifying an organization-defined address for internal mail delivery
- *street_number*: OPTIONAL label string, specifying the number of a building in a street
- *street*: OPTIONAL label string, specifying the name of a street
- *postal_box*: OPTIONAL label string, specifying the number of a postal box
- *town*: OPTIONAL label string, specifying the name of a town
- *region*: OPTIONAL label string, specifying the name of a region
- *postal_code*: OPTIONAL label string, specifying the code that is used by the *country*'s postal service
- *country*: OPTIONAL label string, specifying the name of a country
- *facsimile_number*: OPTIONAL label string, specifying the number at which facsimiles may be received
- *telephone_number*: OPTIONAL label string, specifying the number at which telephone calls may be received
- *electronic_mail_address*: OPTIONAL label string, specifying the electronic address at which electronic mail may be received
- *telex_number*: OPTIONAL label string, specifying the number at which telex messages may be received

Assertions:

- At least one of the attributes shall be specified

4.2.3 approval

An **approval** is an identified confirmation of the quality of the exchange data to which the **approval** pertains.

Attributes:

- *level*: label string, specifying the level of approval. The actual contents of *level* is organization dependent.

EXAMPLE 1 – Examples of *level* are: 'created', 'released for information only', 'checked', 'authorized' and 'agreed by PA officer'.

- *date_time*: reference to a **date_and_time**, specifying the date and time when the approval was made
- *by_person_organization*: reference to a **person_and_organization**, who made the **approval**

4.2.4 ato_aspect

An **ato_aspect** captures the property values of a single **property_class**, obtained for a specific sequence of **network_model_component**s.

Attributes:

- *name*: label string containing the word or words by which the **ato_aspect** is referred to

  EXAMPLE 2 – A name for an **ato_aspect** can be 'charging mode of the batteries'

  EXAMPLE 3 – A name for an **ato_aspect** can be 'temperature of the thermocouples'
- *for_property_class*: a reference to the **property_class_usage** for which this **ato_aspect** expresses the values of the **property_class** referred to by the **property_class_usage**
- *lifetime*: a **lifetime_type**, indicating what the lifetime is for the *scans* of this **ato_aspect**
- *component_sequence*: a reference to a **model_component_sequence**, that specifies for which **network_model_component**s and in what order the values for *for_property_class* are recorded in the *scans*

  NOTE 5 – A **model_component_sequence** may reference a single **network_model**. This allows the **ato_aspect** to record the value of a **property_class** for an entire model – e.g. the reference temperature of the interior of an equipment during a run [DynaWorks].
- *scans*: a list of one or more **scan**s that specify the sampled or derived values for *for_property_class* for the **network_model_component**s of *component_sequence*.
- *security_class*: an OPTIONAL reference to a **security_classification_level**, which specifies the security class for access to the data associated with the **ato_aspect.**

Assertions:

- The order of the elements of *scans* shall be consistent with the *abscissa_sequencing* of *for_property_class.case* when it is specified. If this type is specified then scans shall not contain any **scan_of_derived_values** or shall not contain more than one element.

4.2.5 ato_campaign

An **ato_campaign** is a collection of analysis, test or operation **ato_case**s, performed or to be performed in order to achieve an objective in the context of the development or operation of a product.

NOTE 6 – "ato" stands for "analysis, test or operation"

NOTE 7 – An **ato_campaign** is often an activity within the scope of a particular project.

NOTE 8 – An **ato_campaign** typically pertains to one (major) product variant.

Attributes:

- *of_project*: reference to an **organizational_project** to which this **ato_campaign** belongs
- *name*: label string, specifying the name of the **ato_campaign**
- *description*: text that describes the **ato_campaign**
- *creation_date_and_time*: reference to the **date_and_time** when the **ato_campaign** was created
- *last_modification_date_and_time*: OPTIONAL reference to a **date_and_time**, specifying the latest date and time when the **ato_campaign** or its associated data were modified
- *contact_person_organisation*: a reference to a **person_and_organization**, who acts as a contact on the subject of definition and execution of the **ato_campaign**
- *approvals*: a set of zero or more references to **approval**
- *security_class*: an OPTIONAL reference to a **security_classification_level**, which specifies the security class for access to the data associated with the **ato_campaign**. By default, the **security_class** is indeterminate.

- *application_interpreted_model_schema_name*: name of the AIM EXPRESS SCHEMA of the AP that was used to create the exchange dataset.
- *application_protocol_year*: year number denoting the year of formal publication of the version of the AP used for the creation of the exchange dataset.

Assertions:

- The combination of *of_project* and *name* shall be unique within an exchange dataset

4.2.6 ato_case

An **ato_case** is a definition of a combination of scenario, initial and/or boundary conditions and a **network_model** for the purpose of analysis, test or operation of the product in an **ato_campaign**. An **ato_case** may be decomposed into a sequence of one or more **ato_phase**s (see subclause 4.2.7 on page 4-7).

NOTE 9 – "ato" stands for "analysis, test or operation"

Attributes:

- *of_campaign*: reference to the **ato_campaign** to which this **ato_case** belongs
- *id:* identifier string for the **ato_case**
- *name*: label string which relates the essence of the **ato_case**
- *description*: text that describes the **ato_case**
- *phases*: a set of zero or more references to the **phase**s that constitute the **ato_case**. This is an INVERSE attribute.
- *root_model*: reference to the **network_model** that is the top level supermodel of the model/ submodel tree for the **ato_case**
- *abscissa_class*: reference to a **property_class_usage**, that specifies the property along which the progression of the (analysis, test or operation) case, phase and run are expressed.

  EXAMPLE 4 – `time` is typically the *abscissa_class* for an analysis **ato_case** in which the dynamic behaviour of a product is predicted.

  EXAMPLE 5 – `wavelength` could be the *abscissa_class* for a test **ato_case** in which the emission spectrum of a light source is measured.

  EXAMPLE 6 – `number of cycles` could be the *abscissa_class* for a material fatigue test **ato_case**

  EXAMPLE 7 – `frequency` could be the *abscissa_class* for a frequency sweep in a vibration test **ato_case**
- *abscissa_sequencing :* an OPTIONAL **abscissa_sequencing_type**, optionally indicating how the **abscissa_value** of **scan**s progresses over the list of **scan**s within an **ato_aspect**.

  If no **abscissa_sequencing** is specified, the **abscissa_value**s can progress in any order over the list of **scan**s within an **ato_aspect**.
- *unit_assignments*: a set of one or more **unit_assignment**s, defining in which **unit**s the values are expressed for all **property_class**es associated with the **ato_case**. This attribute is an INVERSE attribute.
- *predefined_aspects*: a set of zero or more **ato_aspect**s referencing **scan**s with property values which are specified before the execution of the **ato_case**. The predefined aspects are applicable to all the phases of the **ato_case.**

  NOTE 10 – The *predefined_aspects* will be typically used to store the following kind of data:
  - calibration curves
  - assigned parameter settings, such as thermostat switching temperatures, environmental parameter values and alarm levels
  - predicted upper and lower profiles for certain properties
- *security_class*: an OPTIONAL reference to a **security_classification_level**, which specifies the security class for access to the data associated with the **ato_case.** By default, the **security_class** is the same as the security class of **of_campaign**. When present, it overrides the security class of **of_campaign**.

Assertions:

- The combination of *of_campaign* and *id* shall be unique within the exchange dataset
- *abscissa_class.property* shall refer to a **property_class_scalar_quantitative** or a **property_class_scalar_descriptive**

Examples:

EXAMPLE 8 – An example of an **ato_case** is an analysis case with the *name* '`summer solstice eclipse and recovery in low earth orbit`'.

EXAMPLE 9 – Another example of an **ato_case** is a test case with the *name* '`Launch vibrations for Ariane 5 spectrum`'.

4.2.7 ato_phase

An **ato_phase** is the next lower decomposition of an analysis, test or operation case (an **ato_case)**. The start and end of an **ato_phase** are marked by predefined events at which (some) parameter settings change. All **ato_phase**s pertaining to one **ato_case** use the same **network_model** and associated model/submodel tree, which is specified by the *root_model* of the **ato_case**.

NOTE 11 – "ato" stands for "analysis, test or operation"

Attributes:

- *of_case:* the **ato_case** that this object specifies a phase of.
- *id:* identifier string
- *name*: label string which relates the essence of the **ato_phase**
- *description*: text that relates the nature of the **ato_phase**
- *predefined_aspects*: a set of zero or more **ato_aspect**s referencing **scan**s with property values which are specified before the execution of the **ato_phase**. The predefined aspects may override predefined aspects of the parent **ato_case.**
- *security_class*: an OPTIONAL reference to a **security_classification_level**, which specifies the security class for access to the data associated with the **ato_phase.** By default, the **security_class** is the same as the security class of the parent **ato_case**. When present, it overrides the security class of the parent **ato_case**.

Assertions:

- The *id* of an **ato_phase** shall be unique within *of_case*.

Notes:

NOTE 12 – If the **ato_phase**s of an **ato_case** should be executed in a particular order, this order shall be determined from the alphanumeric sorting order of the *id*s of the **ato_phase**s.

Examples:

EXAMPLE 10 – An **ato_case** '`summer solstice eclipse and recovery in low earth orbit`' is decomposed into the **ato_phase**s '`initial steady state`', '`eclipse`' and '`recovery`'.

4.2.8 calendar_date

NOTE 13 – The definition of **calendar_date** is taken directly from [STEP-41].

A **calendar_date** is a type of **date** which is identified by a day in a month of a year.

Assertions (in addition to those of **date**)**:**

- *day_component*: the number of the day in the month
- *month_component*: the number of the month in the year

Assertions (in addition to those of **date**)**:**

- None

4.2.9 coordinated_universal_time_offset

NOTE 14 – The definition of **coordinated_universal_time_offset** is taken directly from [STEP-41].

A **coordinated_universal_time_offset** is used to relate a time to coordinated universal time by an offset (specified in hours and minutes) and a direction.

Attributes:

- *hour_offset*: number of hours by which a time is offset from coordinated universal time
- *minute_offset*: OPTIONAL number of minutes by which a time is offset from coordinated universal time
- *sense*: direction of the offset, one item of {`ahead`, `behind`}

Assertions:

- None

4.2.10 cyclic_tabular_function

A **cyclic_tabular_function** is a type of **tabular_function**, which specifies that a tabular function of one variable is periodic.

Attributes (in addition to those of **tabular_function**)**:**

- *period*: **scalar_value**, specifying the period of the function

Assertions (in addition to those of **tabular_function**)**:**

- There shall be only one **property_class_scalar** in the *parameter_properties*, i.e. it is a function of one variable
- The *period* is expressed in the unit of the first element of *parameter_properties*

4.2.11 date

NOTE 15 – The definition of **date** is taken directly from [STEP-41].

A **date** is the identification of a day or week in a year.

The **date** can be either a **calender_date**, an **ordinal_date** or a **week_of_year_and_day_date**.

Attributes:

- *year_component*: the year number in which the **date** occurs

Assertions:

- In this protocol only the **calender_date** specialization shall be valid

  NOTE 16 – The definitions of **ordinal_date** or a **week_of_year_and_day_date** are not given in this document.

4.2.12 date_and_time

NOTE 17 – The definition of **date_and_time** is taken directly from [STEP-41].

A **date_and_time** is a moment of time on a particular day.

Attributes:

- *date_component*: reference to a **date**
- *time_component*: reference to a **local_time**

Assertions:

- None

4.2.13 document

NOTE 18 – The definition of **document** is taken directly from [STEP-41].

A **document** is an unambiguous reference to a formal standard or document that is defined outside this protocol.

Attributes:

- *id*: identifier string
- *name*: label string

- *description*: text, giving a summary description of the document
- *kind*: reference to a **document_type**

Assertions:

- The *id* shall be unique in the exchange dataset

4.2.14 document_type

NOTE 19 – The definition of **document_type** is based on the [STEP-41] equivalent.

A **document_type** is a classification of the sort of data that is described by a **document** is being used to describe in a particular application context.

Attributes:

- *name*: label string, specifying the name of the classification

Assertions:

- None

4.2.15 document_usage_constraint

NOTE 20 – The definition of **document_usage_constraint** is taken directly from [STEP-41].

A **document_usage_constraint** identifies a specific subject area or aspect within a **document** and gives the relevant information or text which applies. The semantics of the reference can be found in the document itself.

Attributes:

- *source*: reference to a **document**
- *subject_element*: label string, specifying the name of one element of the *source*
- *subject_element_value*: text, specifying the contents of the *subject_element*

Examples:

EXAMPLE 11 – A *subject_element* could be '`cleanliness requirements`' with a *subject_element_value* of '`2 through 11, 13, 16, 17, 42 through 73`'.

EXAMPLE 12 – A *subject_element* could be '`Section`' with a *subject_element_value* of '`4.2.5`'.

4.2.16 lifetime_type

A **lifetime_type** is an indication of what the lifetime is for the **scan**s of an **ato_aspect**.

The **lifetime_type** can be one of the following:

- `invariant`: indicates that the values apply to the whole duration of the run
- `sample`: indicates that the values are only valid at the *scan_abscissa* of the **scan**
- `interval`: indicates that the values are valid from the *scan_abscissa* of the current **scan** until the *scan_abscissa* of the next scan.

There are no attributes nor assertions.

4.2.17 list_of_descriptive_values

A **list_of_descriptive_values** is a named list of character strings containing the string values allowed for a **property_class_scalar_descriptive**.

When the *for_property_class* of an **ato_aspect** is a **property_class_scalar_descriptive**, then the values for such a property are stored in the *value*s of the *scans* of the **ato_aspect** as an index indicating the position of the entry in the *entries* of the **list_of_descriptive_values**.

A **list_of_descriptive_values** may contain predefined values which are known at the time when an **ato_case** and/or **ato_phase** is being defined as well as string values that are produced during the execution of a **run**.

Attributes:

- *applicable_for*: reference to a set of one or more **property_class_usage**s to which this **list_of_descriptive_values** applies

- *name*: label string
- *entries*: a list of one or more character strings

  EXAMPLE 13 – The **list_of_descriptive_values** for a **property_class_scalar_descriptive** 'Battery charge mode' can contain the predefined charge modes for batteries: 'off', 'discharge', 'charge' and 'trickle charge'.

  EXAMPLE 14 – A **list_of_descriptive_values** with the *name* 'logbook' can contain logbook entries which are entered during the progression of a test **run**.

  EXAMPLE 15 – A **list_of_descriptive_values** with the *name* 'diagnostic messages' can contain error and status messages generated by an analysis tool during a calculation run.

  EXAMPLE 16 – A **list_of_descriptive_values** with the name 'telemetry' can contain telemetry data strings.

Assertions:

- the attribute *property* of the elements of *applicable_for* shall refer to a **property_class_scalar_descriptive**

4.2.18 list_of_functions

A **list_of_functions** is a named list of function references pointing to the functions for a **property_class_scalar_functional**.

When the *for_property_class* of an **ato_aspect** is a **property_class_scalar_functional**, then the values for such a property are stored in the *value*s of the *scans* of the **ato_aspect** as an index indicating the position of the function reference in the *entries* of the **list_of_functions**.

A **list_of_functions** may contain references to predefined functions which are known at the time when an **ato_case** and/or **ato_phase** is being defined as well as references to functions that are created during the execution of a **run**.

Attributes:

- *applicable_for*: reference to a set of one or more **property_class_usage**s to which this **list_of_functions** applies
- *name*: label
- *entries*: a list of one or more references to **parameterized_function**s

  EXAMPLE 17 – The **list_of_functions** for a **property_class_scalar_descriptive** with the *name* 'Calibration Curves' can contain the references to **parameterized_function**s that represent calibration curves.

Assertions:

- the attribute *property* of the elements of *applicable_for* shall refer to a **property_class_scalar_functional**

4.2.19 local_time

NOTE 21 – The definition of **local_time** is taken directly from [STEP-41].

A **local_time** is a moment of occurrence measured by hour, minutes and second. It represents one instant of time on a 24 hour clock.

It includes the offset to coordinated universal time for the applicable time zone.

Attributes:

- *hour_component*: the hour in day number
- *minute_component*: the minute in hour number
- *second_component*: the second in minute number
- *zone*: a reference to a **coordinated_universal_time_offset**

Assertions:

- None

4.2.20 model_component_sequence

A **model_component_sequence** is a named, ordered collection of **network_model_component**s.

A **model_component_sequence** is referenced by one or more **ato_aspect**s to define for which **network_model_component**s the **ato_aspect** contains **scan**s for a particular property (the **ato_aspect**'s *for_property_class*) and in what order the values for the property are recorded in the **scan**s.

Several types of **network_model_component**s may be present in one **model_component_sequence**.

Attributes:

- *root_model*: reference to a **network_model** that is the highest level supermodel in the model/submodel tree for which this **model_component_sequence** is specified. This information is derived from the definition the elements of *components*.
- *id*: identifier string
- *name*: label string

  EXAMPLE 18 – A **model_component_sequence** could be named '`external surfaces`'.

  EXAMPLE 19 – A **model_component_sequence** could be named '`thermistor-list`'.
- *components*: a list of one or more unique **network_model_component**s

Assertions:

- The combination *root_model* and *id* shall be unique in the exchange dataset
- All *components* shall belong to the *root_model* and its submodel tree

4.2.21 model_represents_product_relationship

A **model_represents_product_relationship** is a relationship between a **network_model** or a **network_model_constituent** and a **product_definition**, which indentifies the product (or part of a product) that is represented by the model constituent.

Attributes:

- *model_constituent*: reference to a **network_model** or to a **network_model_constituent**
- *represented_product:* reference to a **product_definition**

Assertions:

- None

4.2.22 network_model

A **network_model** is a representation, in the form of a network topology, of a product and possibly its environment during analysis, test or operation activities.

The purpose of the **network_model** is to make available a hierarchical decomposition into identified discrete items, in such a way that:

- characteristic properties – e.g. material properties
- predicted properties – e.g. analysis results
- assigned properties – e.g. settings for test parameters
- observed properties – e.g. sensor readings obtained in a test or in operation
- derived properties - e.g. results from statistical operations on sensor readings

can be associated with these items.

NOTE 22 – Many kinds of (numerical) mathematical analysis models can be represented by a network of nodes and the relationships between them: e.g. Lumped Parameter Models, Radiative Surface Models, Finite Element Models, Finite Difference Models and Computational Flow Dynamics Models.

NOTE 23 – Also products under test or in operation can often be represented by a network of nodes and their relationships. Here the nodes would represent the sensors and detectors attached to parts of the product and its environment. The readings for such sensors or detectors would constitute observed (measured) properties, possibly after necessary conversion and/or calibration.

Attributes:

- *id:* identifier string
- *version_id:* identifier string
- *name:* label string
- *description:* text giving a description of the purpose and the characteristics of the model
- *security_class*: OPTIONAL reference to a **security_classification_level**, denoting the security class level for this **network_model**. By default, the security class is indeterminate.
- *constituents:* set of one or more references to **network_model_constituent**s

Assertions:

- The combination *id* and *version_id* shall be unique within the exchange dataset.

4.2.23 network_model_component

A **network_model_component** is an object which provides a generic referencing mechanism for all components of a **network_model**. A **network_model_component** is one of the following:

- **network_model**
- **network_node**
- **network_node_relationship**
- **submodel_usage**
- **network_node_usage**
- **network_node_relationship_usage**

There are no attributes, associations nor assertions.

4.2.24 network_model_constituent

A **network_model_constituent** is an object which provides a generic referencing mechanism for all directly constituent parts of a **network_model**. A **network_model_constituent** is one of the following:

- **network_node**
- **network_node_relationship**
- **submodel_usage**

There are no attributes, associations nor assertions.

4.2.25 network_node

A **network_node** is a generic, atomic item of discretisation in a **network_model**. A **network_node** is a location in the network topology of a **network_model** which is used to represent a physical object or parts thereof. A **network_node** serves as an item at which a **property_class** can be evaluated.

EXAMPLE 20 – A **network_node** can be a node in a thermal lumped parameter model.

EXAMPLE 21 – A **network_node** can be a node in a finite element model.

EXAMPLE 22 – A **network_node** can be a thermocouple or thermistor on a test article.

EXAMPLE 23 – A **network_node** can be a strain gauge.

Attributes:

- *id:* identifier string
- *name:* OPTIONAL label string
- *class*: list of one or more **network_node_class**es, indicating the classes and possibly the subclasses to which the **network_node** belongs. The members in this list are ordered such that the every succeeding member denotes a more specific classification of the preceeding member – the first member of list denotes the most generic, primary classification.

  EXAMPLE 24 – The classes of a thermal lumped parameter node could be:
  {'lumped_parameter_node', 'thermal'}.

  EXAMPLE 25 – The classes of a thermocouple could be:
  {'sensor', 'thermocouple', 'copper_constantan'}.

NOTE 24 – When the classes are presented to the user, it may be useful to present the classes in order of decreasing specialization. The most specific classes will be most significant to the user, because the user will probably be well aware of the most generic classes in the problem domain.

- *security_class*: an OPTIONAL reference to a **security_classification_level**, which specifies the security class for access to the data associated with the **network_node**. By default, the security class is the same as the security class of the parent **network_model**. If present, it overrides the security class of the parent **network_model**.

Assertions:

- The *id* shall be unique within the **network_model** by which the **network_node** is referenced

4.2.26 network_node_class

A **network_node_class** is a specification of a class of **network_node**s, i.e. a named category of nodes which share common characteristics and behaviour.

Attributes:

- *name:* label string, specifying the name of the class

  When applicable, the following *name* shall be used for for the primary *class* of a **network_node**:
  - `thermal_lumped_parameter_node`
  - `finite_element_node`
  - `sensor`

  When applicable, the following *name* shall be used for secondary **network_node_class**es:
  - `thermistor`
  - `thermocouple`
  - `strain_gauge`

Assertions:

- The *name* shall be unique in the exchange dataset

4.2.27 network_node_relationship

A **network_node_relationship** is a representation of a physical relationship between two or more **network_node**s in the network topology of a **network_model**. It serves as an item for which a **property_class** can be evaluated.

EXAMPLE 26 – Examples of such relationships are: the thermal radiative exchange factors between two thermal-radiative faces; the linear conductors, radiative couplings and mass flow links between two thermal lumped parameter nodes; a finite element as a relationship between its vertex nodes.

Attributes:

- *id:* identifier string
- *name*: an OPTIONAL label string
- *class*: list of one or more **network_node_relationship_class**es, indicating the classes and possibly the subclasses to which the **network_node_relationship** belongs. The members in this list are ordered such that the every succeeding member denotes a more specific classification of the preceeding member – the first member of list denotes the most generic, primary classification.

  EXAMPLE 27 – The classes of a thermal radiative coupling could be:
  {`'thermal_radiative_coupling'`}.

  EXAMPLE 28 – The classes of a linear 3D tetrahedron finite element could be:
  {`'finite_element'`, `'volume_3d_tetrahedron_linear'`}.

  NOTE 25 – When the classes are presented to the user, it may be preferred to present the classes in order of decreasing specialization. The most specific classes will be most significant to the user, because the user will probably be well aware of the most generic classes in the problem domain.

- *security_class*: an OPTIONAL reference to a **security_classification_level**, which specifies the security class for access to the data associated with the **network_node_relationship**. By default, the security class is the same as the security class of the parent **network_model**. If present, it overrides the security class of the parent **network_model**.
- *nodes:* list of two or more references to **network_node** or **network_node_usage**

  NOTE 26 – When the represented relationship is directional, the positive direction shall be from a **network_node** in the *nodes* list with a lower index to one with a higher index.

  NOTE 27 – A relationship for a node with itself – e.g. an extended viewfactor or a radiative exchange factor for a thermal-radiative face – is possible by referencing the same node twice.

  NOTE 28 – If an element from the *nodes* list is a reference to a **network_node_usage** then the **network_node_relationship** specifies the existence of an inter-model relationship between two or more nodes, from which at least one node belongs to a submodel of the model for which the **network_node_relationship** is specified. This mechanism enables the specification of relationships that exist only in the context of a supermodel.

  EXAMPLE 29 – An example of such an inter-model relationship is the representation of a thermal-radiative coupling between two units that are both modelled as submodels. Such a coupling would reference thermal interface nodes of the models for each unit.

Assertions:

- The *id* shall be unique within the **network_model** for which the **network_node_relationship** is specified
- If an element from the *nodes* list is a reference to a **network_node** then it shall be a reference to a **network_node** which belongs to the **network_model** for which the **network_node_relationship** is specified
- If an element from the *nodes* list is a reference to a **network_node_usage** then the *used_submodel* of the **network_node_usage** shall belong to model/submodel tree of the **network_model** for which the **network_node_relationship** is specified, i.e. a **network_node_relationship** can not reference **network_node**s which belong to a supermodel of the **network_model** for which the **network_node_relationship** is specified.

4.2.28 network_node_relationship_class

A **network_node_relationship_class** is a specification of a class of **network_node_relationship**s, i.e. a named category of node relationships which share common characteristics and behaviour.

Attributes:

- *name:* label string, specifying the name of the class

  When applicable, the following *name* shall be used for the primary *class* of a **network_node_relationship**:
  - `electrical_conductor`
  - `finite_element`
  - `geometric_viewfactor`
  - `radiative_exchange_factor`
  - `thermal_radiative_coupling`
  - `thermal_conductor`
  - `mass_flow_link`

  When applicable, the following *name* (derived from the element topologies in the [STEP-104] **structural_response_representation_schema**) shall be used for the *class* of a **network_node_relationship** which belongs to the primary **network_node_relationship_class** with the *name* `'finite_element'`:
  - `curve_3d_linear`
  - `surface_2d_linear`
  - `curve_3d_quadratic`
  - `surface_2d_quadratic`
  - `curve_3d_cubic`

- surface_2d_cubic
- surface_3d_triangle_linear
- volume_2d_triangle_linear
- surface_3d_triangle_quadratic
- volume_2d_triangle_quadratic
- surface_3d_triangle_cubic
- volume_2d_triangle_cubic
- surface_3d_quadrilateral_linear
- volume_2d_quadrilateral_linear
- surface_3d_quadrilateral_quadratic
- volume_2d_quadrilateral_quadratic
- surface_3d_quadrilateral_cubic
- volume_2d_quadrilateral_cubic
- volume_3d_hexahedron_linear
- volume_3d_hexahedron_quadratic
- volume_3d_hexahedron_cubic
- volume_3d_wedge_linear
- volume_3d_wedge_linear
- volume_3d_wedge_quadratic
- volume_3d_tetrahedron_linear
- volume_3d_tetrahedron_linear
- volume_3d_tetrahedron_quadratic
- volume_3d_pyramid_linear
- volume_3d_pyramid_linear
- volume_3d_pyramid_quadratic

Assertions:

- The *name* shall be unique in the exchange dataset

4.2.29 network_node_relationship_usage

A **network_node_relationship_usage** is a specification of a reference to a **network_node_relationship** in a particular **submodel_usage**. Its purpose is to enable the referencing in a **model_component_sequence** of a **network_node_relationship** used in a particular submodel.

Attributes:

- *used_submodel*: reference to a **submodel_usage**
- *relationship:* reference to a **network_node_relationship**

Assertions:

- The *relationship* shall belong to the *submodel* of the *used_submodel*

4.2.30 network_node_usage

A **network_node_usage** is a specification of a reference to a **network_node** in a particular **submodel_usage**. Its purpose is to enable the referencing in a **model_component_sequence** of a **network_node** used in a particular submodel.

Attributes:

- *used_submodel*: reference to a **submodel_usage**
- *node:* reference to a **network_node**

Assertions:

- The *node* shall belong to the *submodel* of the *used_submodel*

4.2.31 organization

NOTE 29 – The definition for **organization** is taken directly from [STEP-41].

An **organization** is an administrative structure.

Attributes:

- *id*: OPTIONAL identifier string, by which the **organization**'s individuality may be deduced.
- *name*: label string, by which the **organization** is named
- *description*: text, describing the nature of the **organization**

Assertions:

- If specified, the *id* shall be unique in the exchange dataset.

4.2.32 organizational_address

NOTE 30 – The definition for **organizational_address** is taken directly from [STEP-41].

An **organizational_address** is a type of **address** where **organization**s are located and/or can be contacted.

Attributes:

- *organizations*: set of references to one or more **organization**s
- *description*: text that relates the nature of the **organizational_address**

Assertions:

- None

4.2.33 organizational_project

NOTE 31 – The definition for **organizational_project** is taken directly from [STEP-41].

An **organizational_project** is a project for which one or more **organization**s are responsible.

Attributes:

- *name*: label string, specifying the name of the project
- *description*: text, that relates the nature of the project
- *responsible_organizations*: set of references to one or more **organization**s

Assertions:

- None

4.2.34 parameterized_function

A **parameterized_function** is an abstract object and captures in a generic way a function of one or more parameters, which are physical quantities.

A **parameterized_function** can be one of the following:

- **parameterized_function_with_formula**
- **polynomial_function**
- **tabular_function**

Attributes:

- *name*: label string, specifying the name of the function
- *description*: text, describing use and validity of the function
- *parameter_properties*: list of zero or more references to **property_class_scalar_quantitative** that define the properties of the function (input) parameters.
- *result_property*: reference to the **property_class_scalar_quantitative**, that defines the property of the result of the function
- *source*: OPTIONAL reference to a **document_usage_constraint**, which specifies the source of the function definition
- *definition_timestamp*: OPTIONAL reference to a **date_and_time**, specifying the date and time when the function was defined

Assertions:

- None

4.2.35 parameterized_function_with_formula

A **parameterized_function_with_formula** is a type of **parameterized_function** in which the function is described by a character string representing a formula.

Attributes (in addition to those of **parameterized_function**)**:**

- *formula*: text, containing the formula that describes the function

  NOTE 32 – The format of the formula is proprietary to the preprocessor (associated with the *creator_tool_or_facility*) that created it. In general the formula serves as "for information only" data and only specifically adapted postprocessors will be able to interprete it on reception.
- *language*: OPTIONAL label string, denoting the language in which the formula is expressed

  EXAMPLE 30 – An example of *language* could be `'Dynaworks'`
- *creator_tool_or_facility*: OPTIONAL label string, denoting the tool or facility that created the formula

  EXAMPLE 31 – An example of *creator_tool_or_facility* could be `'Dynaworks 3.1'`

Assertions (in addition to those of **parameterized_function**)**:**

- None

4.2.36 person

NOTE 33 – The definition for **person** is taken directly from [STEP-41].

A **person** is an individual human being.

Attributes:

- *id*: identifier string, by which the **person** may be identified
- *last_name*: OPTIONAL label string
- *first_name*: OPTIONAL label string
- *middle_names*: OPTIONAL list of one or more label strings
- *prefix_titles*: OPTIONAL list of one or more label strings
- *suffix_titles*: OPTIONAL list of one or more label strings

Assertions:

- The *id* shall be unique in the exchange dataset.
- At least one from *last_name* and *first_name* shall be specified.

4.2.37 person_and_organization

NOTE 34 – The definition for **person_and_organization** is taken directly from [STEP-41].

A **person_and_organization** is a person in an organization.

Attributes:

- *the_person*: reference to a **person**
- *the_organization*: reference to an **organization**

Assertions:

- None

4.2.38 personal_address

NOTE 35 – The definition for **personal_address** is taken directly from [STEP-41].

A **personal_address** is a type of **address** where people reside and/or can be contacted.

Attributes:

- *people*: set of references to one or more **person**s
- *description*: text that relates the nature of the **personal_address**

Assertions:

- None

4.2.39 polynomial_function

A **polynomial_function** is a type of **parameterized_function** that implements the function definition in the form of a polynomial function of one, two or three variables, with a maximum degree of ten.

The bases to be used for the definition of the polynomial function are given below:

Base for a polynomial with one parameter x_1 :

$$(1, x_1, x_1^2, x_1^3, x_1^4, x_1^5, \ldots) \quad \text{(eq.4-1)}$$

Base for a polynomial with two parameters x_1, x_2 :

$$(1, x_1, x_2, x_1^2, x_1x_2, x_2^2, x_1^3, x_1^2x_2, x_1x_2^2, x_2^3, \ldots) \quad \text{(eq.4-2)}$$

Base for a polynomial with three parameters x_1, x_2, x_3 :

$$(1, x_1, x_2, x_3, x_1^2, x_1x_2, x_1x_3, x_2^2, x_2x_3, x_3^2, \quad \text{(eq.4-3)}$$
$$x_1^3, x_1^2x_2, x_1^2x_3, x_1x_2^2, x_1x_2x_3, x_1x_3^2, x_2^3, x_2^2x_3, x_2x_3^2, x_3^3, \ldots)$$

Attributes (in addition to those of **parameterized_function**):

- *degree*: positive integer defining the degree of the polynomial
- *coefficients*: list of real values, which specify all coefficients, ordered as in the base equations above
- *lower_domain_bounds*: list of three OPTIONAL real values, that specify the lower limits of the domain for which the function is valid, for each of the function parameters (specified by *parameter_properties* of **parameterized_function**)
- *upper_domain_bounds*: list of three OPTIONAL real values, that specify the lower limits of the domain for which the function is valid, for each of the function parameters (specified by *parameter_properties* of **parameterized_function**)

Assertions (in addition to those of **parameterized_function**):

- The number of *parameter_properties* of the **parameterized_function** shall be no more than 3 – i.e. a function of at most 3 variables
- The *degree* shall be in the range from 1 to 10
- The *coefficients* are expressed in units consistent with the actual **unit_assignment**s for the *parameter_properties* and the *result_property* of the **polynomial_function**

Notes:

NOTE 36 – The lower and upper bounds are included in the domain of validity of the parameters.

NOTE 37 – When not defined, a lower bound equals: –infinity.

NOTE 38 – When not defined, an upper bound equals: +infinity.

4.2.40 product

A **product** is an identification and textual description of a physically realisable object that is produced by a natural or artificial process.

Attributes:

- *id:* identifier string
- *name*: label string containing the word or group of words by which the **product** is referred to.
- *description*: text that relates the nature of the **product**.
- *frame_of_reference*: one or more references to a **product_context**.

Assertions:

- The *id* shall be unique in the exchange dataset.

Examples:

EXAMPLE 32 – An example of an *id* is a part number.

4.2.41 product_context

A **product_context** collects information on the engineering or manufacturing perspective in which the product data are defined. This context may affect the meaning and usage of the product data.

Attributes:

- *name*: label string containing the name of the context in which the product data exists.
- *discipline_type*: label string which identifies the discipline to which the product data belong.

Assertions:

- For STEP-NRF exchange datasets the *discipline_type* shall be 'definition and execution of analysis' or 'test and operations using network models'.

4.2.42 product_definition

A **product_definition** is an identification of a characterisation of a product in a particular application context.

Attributes:

- *id:* identifier string
- *description:* text that relates the nature of the product.
- *frame_of_reference*: reference to a **product_definition_context**.

Assertions:

- The *id* shall be unique in the *frame_of_reference*.

Notes:

NOTE 39 – A **product**'s physical design may be one **product_definition** while the functional design of the same **product** may be a different **product_definition**.

4.2.43 product_definition_context

A **product_definition_context** collects information on the stage in the product life cycle to which the product data belongs. This context may affect the meaning and usage of the product data.

Attributes:

- *name*: label string containing the name of the context in which the product data exists.
- *life_cycle_stage*: label string which identifies the life cycle stage to which the product data belong.

Examples:

EXAMPLE 33 – Typical values for *life_cycle_stage* are: 'conceptual design', 'preliminary design', 'detailed design'.

4.2.44 product_next_assembly_usage_relationship

A **product_next_assembly_usage_relationship** is an identified association between two **product_definition**s. It specifies that one **product_definition** is the next level higher assembly which uses the other **product_definition** as a constituent. This relationship is used to specify an assembly tree, which is also often called product tree or manufacturing tree.

Attributes:

- *id*: identifier
- *assembly*: reference to a **product_definition**
- *constituent*: reference to a **product_definition**

Assertions:

- The combination *id* and *assembly* shall be unique in the exchange dataset.

- The relationship shall form an acyclic graph, i.e. no circular references to **product_definition** instances are allowed

4.2.45 product_version

A **product_version** is an identified grouping of **product_definition**s, that together define the version of a product. A version differs from other versions of a **product** in some significant way. However, the difference is insufficient to regard the version as a different **product**.

NOTE 40 – The rules for differentiating between the case in which the changes to a product are great enough to justify the definition of a new product and the case in which the changes only require the definition of a new version of a product are enterprise specific. Such rules are not specified in the scope of this protocol.

Attributes:

- *of_product*: reference to a **product**
- *id:* identifier string
- *description*: textual description of the product version
- *definitions*: references to one or more **product_definition**s

Assertions:

- The combination *of_product* and *id* shall be unique in the exchange dataset.

Notes:

WORKING NOTE 5 – The current definition in [STEP-41] for **product_definition** allows referencing only one version of the product (**product_definition_formation**).

Examples:

EXAMPLE 34 – The version identifier '`v3.11`' for a part is an example of an *id*.

4.2.46 property_class

A **property_class** is an abstract object which enables a generic and efficient way for the referencing of a scalar or tensor of characteristic, predicted, assigned, observed or derived property class definition. Property classes can optionally denote a role. The values of the property can be quantitative, descriptive or a functional.

A **property_class** can be a **property_class_scalar** or a **property_class_tensor**.

WORKING NOTE 6 – It would be very well possible to extend the protocol to handle properties that cannot be expressed as a simple value. This would require the definition of an additional specialization of **property_class_scalar** (e.g. a **property_class_scalar_contained**), which would have the following attributes:
– *MIME_type*
– *MIME_subtype*
– *encoding* (such as `octet`, `uuencoded`, `base64`, `binhex-4`)
– *container* (for documents stored inline)
– *url* (Uniform Resource Locator for externally stored documents)
However, a more detailed definition of these objects is outside the scope of the current version of the protocol.

4.2.47 property_class_scalar

A **property_class_scalar** is a **property_class** for a scalar property, which can be an element in a **property_class_tensor**. It can be a **property_class_scalar_descriptive**, **property_class_scalar_functional** or **property_class_scalar_quantitative**.

Attributes:

- *name:* reference to a **property_name** that specifies the kind of quantity dealt with by the **property_class_scalar**
- *symmetry:* OPTIONAL reference to **property_symmetry**

4.2.48 property_class_scalar_descriptive

A **property_class_scalar_descriptive** is a type of **property_class_scalar** for which the value is expressed as a string selected from a **list_of_descriptive_values**.

Attributes (in addition to those of **property_class_scalar**):

- *value_container*: a set of one or more **list_of_descriptive_value**s that contains the actually possible string values. This attribute is a DERIVE attribute.

Assertions (in addition to those of **property_class_scalar**):

- none

Examples:

EXAMPLE 35 – 'battery charge mode' could be the *name* of a **property_class_scalar_descriptive**, which uses a **list_of_descriptive_values** containing the predefined charge modes for batteries: 'off', 'discharge', 'charge' and 'trickle charge'.

EXAMPLE 36 – 'Thermal_Balance_Test_Logbook' could be the *name* of a **property_class_scalar_descriptive**, which uses a **list_of_descriptive_values** containing the logbook entries made during the progression of a **run**. The values for such a property (recorded in the *scans* of an **ato_aspect**) would typically be associated with the whole model for which the test is executed – i.e. an **ato_aspect** with a *component_sequence* containing only one component: the root **network_model**.

4.2.49 property_class_scalar_functional

A **property_class_scalar_functional** is a type of **property_class_scalar** for which the value is a reference to a function.

Attributes (in addition to those of **property_class_scalar**):

- *function_container*: a set of one or more **list_of_function**s that contains the references to the actual functions, which are possible for the **property_class_scalar_functional** . This attribute is an DERIVE attribute.

Assertions (in addition to those of **property_class_scalar**):

- none

Examples:

EXAMPLE 37 – A **property_class_scalar_functional** with the name 'thermocouple calibration curves' could be used to specify the actual calibration curves for thermocouples at the time of definition of a test. In the *predefined_aspects* of an **ato_case** or **ato_phase** an **ato_aspect** of the **property_class_scalar_functional** would be referenced with *scans* indicating to which thermocouples (specified as **network_node**s in a **model_component_sequence**) a particular calibration curve applies. The calibration curves themselves would be specified by **polynomial_function**s of degree 2 with one *parameter_properties* with the *name* 'voltage' and a *result_property* with the name 'temperature'.

4.2.50 property_class_scalar_quantitative

A **property_class_scalar_quantitative** is a type of **property_class_scalar** for which the value is expressed as a numerical value.

Attributes (in addition to those of **property_class_scalar**):

- none

Assertions (in addition to those of **property_class_scalar**):

- none

Examples:

EXAMPLE 38 – Temperature can be specified by a **property_class_scalar_quantitative** with *name* 'temperature' and no *roles*.

EXAMPLE 39 – Ambient temperature can be specified by a **property_class_tensor** with *name* 'temperature' and *roles* {'ambient'}.

EXAMPLE 40 – The upper qualification limit for pressure can be specified by a **property_class_scalar_quantitative** with *name* 'pressure' and *roles* {'limit', 'qualification', 'upper'}.

EXAMPLE 41 – 'battery_charge_mode' can *not* be a **property_class_scalar_quantitative**, because it cannot be evaluated numerically.

4.2.51 property_class_tensor

A **property_class_tensor** is the specification of a class of a tensor of scalar properties.

NOTE 41 – The scalar properties, members of the tensor, are specified through the object **scalar_in_tensor**.

NOTE 42 – In the case of a sparse tensor, some of the possible members may be left without any **property_class_scalar** assigned.

Attributes:

- *name*: reference to a **property_name** that specifies the kind of quantity dealt with by the **property_class_tensor**.
- *tensor_order*: integer number, defining the tensor order:
  - 1 for a vector
  - larger than 1 for a higher order tensor
- *dimensionality*: an integer number indicating the dimensionality of the tensor
- *members*: a set of one or more **scalar_in_tensor**s, specifying the property classes that are gathered in the tensor. This attribute is an INVERSE attribute.

Assertions:

- The combination of *tensor_order* and *dimensionality* shall be one of the following:
  - *tensor_order* = 1, *dimensionality* ≥ 2
  - *tensor_order* = 2, *dimensionality* = 2 or 3
  - *tensor_order* = 4, *dimensionality* = 2 or 3

EXAMPLE 42 – The *members* of a three-dimensional vector a are ordered as a_1, a_2, a_3.

EXAMPLE 43 – The *members* of a 4th order two-dimensional tensor A_{ijkl} are ordered as:

$a_{1,1,1,1}$, $a_{1,1,1,2}$, $a_{1,1,2,1}$, $a_{1,1,2,2}$, $a_{1,2,1,1}$, $a_{1,2,1,2}$, $a_{1,2,2,1}$, $a_{1,2,2,2}$,
$a_{2,1,1,1}$, $a_{2,1,1,2}$, $a_{2,1,2,1}$, $a_{2,1,2,2}$, $a_{2,2,1,1}$, $a_{2,2,1,2}$, $a_{2,2,2,1}$, $a_{2,2,2,2}$

4.2.52 property_class_usage

A **property_class_usage** is the characterization of the usage of a **property_class** in an **ato_case**. It specifies the meaning of the property in the context of the **ato_case**.

Attributes:

- *of_case*: refers to the considered **ato_case**;
- *property*: refers to a **property_class** that is considered during the case;
- *meaning*: specifies a list of one or more **property_role**s that qualifies the meaning of the property in the context of the **ato_case**. The members in this list are ordered such that the every succeeding member denotes a more specific classification of the preceeding member - the first member of list denotes the most generic, primary role.

  NOTE 43 – Sometimes it may be diffult to determine the ordering of the roles, e.g. in 'upper design limit temperature', is 'upper' a specialization of the roles 'design limit', or is 'design' a specialization of the roles 'upper limit'? A rule-of-thumb solution can often be found by expressing the role in a natural language. Most natural languages developed an intuitive hierarchy for the ordering of terms that denoting specialization. In English, the ordering is from specific to generic. For this example, this means that 'upper' denotes a specialization of 'design', which denotes a specialization of the role 'limit'.

NOTE 44 – When the roles are presented to the user, it may be preferred to present the roles in order of decreasing specialization. The most specific roles will be most significant to the user, because the user will be probably well aware of the most generic roles in the problem domain.

4.2.53 property_name

A **property_name** is a character string that specifies the kind of quantity that a **property_class_scalar** or a **property_class_tensor** covers.

Attributes:

- *name*: label string, specifying the name

  When applicable, a *name* from the following list shall be used:

  - absorptance
  - acceleration
  - amount_of_substance
  - angle
  - cartesian_position
  - cartesian_velocity
  - charge
  - count
  - density
  - electric_current
  - electric_conductance
  - electric_potential
  - electric_resistance
  - emittance
  - flux
  - force
  - frequency
  - heat_capacity
  - irradiance
  - length
  - luminous_intensity
  - mass
  - moment_of_inertia
  - number
  - plane_angle
  - polar_position
  - power
  - pressure
  - ratio
  - reflectance
  - relative_humidity
  - roughness
  - solid_angle
  - specularity
  - spherical_position
  - strain
  - stress
  - temperature
  - thermal_conductance
  - thermodynamic_temperature
  - time
  - torque

- velocity
- voltage
- volume
- wavelength
- work

Assertions:

- The *name* must be unique within the exchange dataset

4.2.54 property_role

A **property_role** is the specification of a role of a **property_class** (a **property_class_scalar** or a **property_class_tensor**) as such or the specification of a role of a **property_class_scalar** within a **property_class_tensor**.

Attributes:

- *name*: label string, specifying the role name

  When applicable, the following *name* shall be used:

  - absolute
  - alarm
  - ambient
  - design
  - displacement
  - distance
  - disturbed
  - dynamic
  - hemi_spherical
  - infrared
  - initial
  - limit
  - linear
  - lower
  - maximum
  - mean
  - measured
  - minimum
  - negative
  - nominal
  - non_linear
  - normal
  - phi
  - positive
  - predicted
  - relative
  - residual
  - solar
  - static
  - switch_on
  - switch_off
  - theta
  - total
  - upper
  - worst_case
  - x
  - y

- z

Assertions:

- the *name* shall be unique within the exchange dataset

Examples:

EXAMPLE 44 – 'ambient' can be the *name* of a the **property_role** referenced by a **property_class_scalar** with *name* 'temperature'.

4.2.55 property_symmetry

A **property_symmetry** specifies the behaviour of the property assigned to a **network_node_relationship** (and **network_node_relationship_usage**s which reference it) between two **network_nodes**. Its purpose is to inform an application that uses the results data how the value of the inverse relationship can be derived – this prevents the storage of redundant data. There are three possibilities:

1) The relationship is symmetrical. This means that the value of the property of the relationship between nodes i and j ($\mathbf{P}_{ij}$) is the same as the value of the same property of the same relationship between nodes j and i ($\mathbf{P}_{ji}$): $\mathbf{P}_{ji} = \mathbf{P}_{ij}$.
2) The relationship is antisymmetrical. This means that the value of the property of the relationship between nodes i and j ($\mathbf{P}_{ij}$) is the opposite of the value of the same property of the same relationship between nodes j and i ($\mathbf{P}_{ji}$): $\mathbf{P}_{ji} = -\mathbf{P}_{ij}$.
3) Providing information on the symmetry is meaningless. This means that the value of the property of the relationship between nodes i and j ($\mathbf{P}_{ij}$) does not provide a simple way to derive the value of the same property of the same relationship between nodes j and i ($\mathbf{P}_{ji}$), or that the reverse relationship is even meaningless. In this case the *symmetry* information is undefined.

   EXAMPLE 45 – The **property_class_scalar** with the *name* 'electrical resistance' of a **network_node_relationship** that represents a resistor between *nodes* 1 and 2 is symmetrical: the values of the resistances $\mathbf{R}_{1,2}$ and $\mathbf{R}_{2,1}$ are the same.

   EXAMPLE 46 – The **property_class_scalar** with the *name* 'heat flow' for a **network_node_relationship** that represents a thermal conductor between *nodes* 1 and 2 is antisymmetrical: the values of the heatflows $\mathbf{q}_{1,2}$ and $\mathbf{q}_{2,1}$ are each other's opposites: $\mathbf{q}_{1,2} = -\mathbf{q}_{2,1}$.

A **property_symmetry** is one item from the list {symmetrical, antisymmetrical}.

There are no attributes nor assertions.

4.2.56 run

A **run** is a named and time-stamped collection of references (through **ato_aspect**s) to results data that together establish an executed analysis, test or operation **run**.

Attributes:

- *of_phase:* reference to the **ato_phase** for which this executed **run** produces results.
- *id:* identifier string, giving a unique identification code to a **run** instance
- *name*: label string containing the word or group of words by which the **run** is referred to.
- *description*: text that relates the nature of the **run**.
- *timestamp*: calendar **date_and_time** marking the start of the **run**
- *creator_tool_or_facility*: OPTIONAL label string, denoting the tool or facility that created the **run** results
- *aspects*: one or more references to **ato_aspect**s, for which property values are produced in the run

Assertions:

- The combination of *of_phase* and *id* shall be unique in the exchange dataset.

4.2.57 run_sequence

A **run_sequence** is a sequence of **run**s, which belong to a single **ato_case**.

Attributes:

- *of_case:* reference to the **ato_case** for which this executed **run_sequence** produces results.
- *id:* identifier string
- *name*: label string containing the word or group of words by which the **run_sequence** is referred to.
- *description*: text that relates the nature of the **run_sequence**.
- *runs*: a list of one or more **run**s that belong to this **run_sequence**

Assertions:

- The combination of *of_case* and *id* shall be unique within the exchange dataset
- The **ato_phase**s referenced by *runs* through *of_phase* shall all belong to *of_case* (through its *phases* attribute)

Examples:

EXAMPLE 47 – A **run_sequence** with the *name* 'ESATAN analysis of summer solstice eclipse and recovery in low earth orbit' represents the **ato_case** 'summer solstice eclipse and recovery in low earth orbit', and can be decomposed in the **run**s 'initial steady state', 'eclipse' and 'recovery'.

4.2.58 scalar_in_tensor

A **scalar_in_tensor** identifies one **property_class_tensor** that is used as a member of a **property_class_tensor** and its role in the tensor.

- *tensor:* reference to the considered **property_class_tensor**
- *scalar:* reference to the considered **property_class_scalar**
- *position*: specifies the position of the **property_class_scalar** within the **property_class_tensor.** See 4.2.51 for the definition of the order of the members of a **property_class_tensor**
- *roles:* a list of one or more **property_role**s, denoting the role of the ***scalar*** in the *tensor* . The items in this list are ordered in such a way that the first item specifies the most generic, primary role and that each further item in the list qualifies its preceeding item.

  NOTE 45 – the same **property_class_scalar** may be used several times within a **property_class_tensor**, with different roles: a three dimensional cartesian displacement can be represented by a **property_class_tensor** with a *tensor_order* of 1 (indicating a vector) and a *dimensionality* of 3, where each of the elements of the vector is a **property_class_scalar** with *name* 'length' and the respective *role*s 'x', 'y' and 'z'.

4.2.59 scalar_value

A **scalar_value** is a number that represents the value of a **property_class_scalar**, which may be a member of a **property_class_tensor**. This number can be:

- the numerical value for a quantitative property (of a **property_class_scalar_quantitative**)
- an index value for a descriptive property (of a **property_class_scalar_descriptive**) which specifies the position of the character string representation of the descriptive value in the associated **list_of_descriptive_values**.
- an index value for a functional property (of a **property_class_scalar_functional**) which specifies the position of the function pointer in the associated **list_of_functions**.

Assertions:

- If the **property_class_scalar** is a **property_class_scalar_descriptive**, the **scalar_value** shall be a valid index to an item in the **list_of_descriptive_value**s, specified by *value_container*
- If the **property_class_scalar** is a **property_class_scalar_functional**, the **scalar_value** shall be a valid index to an item in the **list_of_function**s, specified by *function_container*

4.2.60 scan

A **scan** is a collection of values for the **property_class** and **network_model_component**s specified by *of_property_class* and *component_sequence* in an **ato_aspect**, sampled at one particular value for the *abscissa_class* or derived by a specific **scan_derivation_procedure**.

A **scan** specializes into **scan_of_sampled_values** and **scan_of_derived_values**.

Attributes:

- *of_aspect*: the **ato_aspect** for which the **scan** collects a set of values. This is an INVERSE attribute.
- *abscissa_value*: an OPTIONAL **scalar_value**, that specifies the value of the property defined by *abscissa_class* for which the **scan** is sampled or derived. The *abscissa_class* is specified by the **ato_case** to which the **scan** pertains, through the objects **ato_case**, **ato_phase**, **ato_aspect** and **run**.
- *mask*: an OPTIONAL reference to a **scan_mask**, that may specify for which *components* of the **model_component_sequence** a property value (for a scalar) or list of property values (for a vector or higher order tensor) is present. The purpose of the mask is to allow for efficient storage of incomplete scans of property values.
  If no **scan_mask** is specified, then property values are given for all *components* listed in the *component_sequence* of the **ato_aspect**.
- *values*: a list of one or more **scalar_value**s

Assertions:

- The number of *values* shall be as defined below:
  - No *mask* specified and the **ato_aspect**'s *for_property_class* is a **property_class_scalar**: the number of *components* in the *component_sequence* of the **ato_aspect**
  - No *mask* specified and the **ato_aspect**'s *for_property_class* is a **property_class_tensor**: the product of the number of *components* and the number of *members* in the **property_class_tensor**
  - *mask* specified and the **ato_aspect**'s *for_property_class* is a **property_class_scalar**: the number of *components* implied by *defined_ranges* in *mask*
  - *mask* specified and the **ato_aspect**'s *for_property_class* is a **property_class_tensor**: the product of the number of *components* implied by *defined_ranges* in *mask* and the number of *members* in the **property_class_tensor**
- The ordering of the *values* list is as follows:
  - The property values are in the same order as the *components* in the *component_sequence* of the **ato_aspect**
  - If the *for_property_class* is a **property_class_tensor** then the ordering within the property value list is the same as for the tensor *members*, i.e. such that the highest dimension changes fastest.

4.2.61 scan_derivation_procedure

A **scan_derivation_procedure** is a character string denoting the procedure used to derive a **scan_of_derived_values** from a collection of **scan_of_sampled_values**.

Attributes:

- *name*: a label string denoting the procedure.
  When applicable, the following *name* shall be used:
  - `minimum`
  - `maximum`
  - `mean`
  - `standard_deviation`
  - `variance`
  - `median`
  - `mode`
  - `sum`

Assertions:

- the *name* shall be unique within the exchange dataset

4.2.62 scan_mask

A **scan_mask** is a list that specifies for which **network_model_component**s property values are available in one or more **scans**. The purpose of this mask is to allow for efficient storage of incomplete (sparse) data.

More than one **scan** can reference the same **scan_mask**.

Attributes:

- *defined_ranges*: a list of pairs of indices. Each pair indicates the beginning and ending position of a range within the *component_sequence* list in the **ato_aspect**. Any **network_model_component** that falls within these ranges will have a *values* entry in a **scan** that references this **scan_mask**.

Assertions:

- the *defined_ranges* shall not overlap: no **network_model_component** may be referenced in more than one *defined_range* within a **scan_mask**

Examples:

EXAMPLE 48 – In a thermal balance test **run**, temperatures are measured for 200 thermocouples. These thermocouples are listed in the *component_sequence* of the 'thermocouple temperatures' **ato_aspect**. During a certain part of this test **run**, temperatures are only available for 50 thermocouples – the first 20 and the last 30 in the list.
For the scans in this phase, the results can be stored in two ways:
1) no **scan_mask** is assigned for the **scans** in this phase, and **scalar_value**s are stored as indeterminate values in the **scan**s for the 21rd to 170th thermocouple.
2) a **scan_mask** is defined containing the ranges (1:20) and (171:200), and the **scan**s for this phase reference this **scan_mask** and store only a list of 50 **scalar_value**s.
In terms of storage efficiency, the second approach is much more efficient.

4.2.63 scan_of_derived_values

A **scan_of_derived_values** is a type of **scan** that captures a recording of the values for the **property_class** and **network_model_component**s considered in an **ato_aspect**, derived by a specific procedure.

NOTE 46 – Typically, a **scan_of_derived_values** holds the results of statistical processing of a series of **scan_of_sampled_values**.

Attributes (in addition to those of **scan):**

- *derivation_procedure:* a reference to **scan_deriviation_procedure** denoting the procedure which has been used to obtain the values.

  EXAMPLE 49 – A **scan_of_derived_values** containing the mean temperature for every **network_model_component** (averaged over a whole run) can refer to the **scan_derivation_procedure** with the *name* 'mean'.
- *abscissa_start*: an OPTIONAL **scalar_value**, expressing the value of the *abscissa_class* of the start of the interval where the *procedure* has been applied.
- *abscissa_end*: an OPTIONAL **scalar_value**, expressing the value of the *abscissa_class* of the end of the interval where the *procedure* has been applied.

  NOTE 47 – Typically, the *abscissa_start* and *abscissa_end* will be used to indicate an interval of **scan_of_sampled_values** on which some statistical processing has been performed.

  NOTE 48 – If *abscissa_start* is unspecified, then the *procedure* has been applied with considering all the **scan**s of the **run**, that have an *abscissa_value* less or equal than *abscissa_end*.

  NOTE 49 – If *abscissa_end* is unspecified, then the *procedure* has been applied with considering all the **scan**s of the **run**, that have an *abscissa_value* greater or equal than *abscissa_start*.

Assertions (in addition to those of **scan):**

- The combination of *procedure*, *abscissa_start* and *abscissa_end* shall be unique within an **ato_aspect**.

EXAMPLE 50 – A series of **scan_of_derived_values**, with specified *abscissa_start* and *abscissa_end* and a **scan_derivation_procedure** named `'mean'` can be used to store the results of low-pass filter statistical processing, which averages the values within certain intervals.

4.2.64 scan_of_sampled_values

A **scan_of_sampled_values** is a type of **scan** that captures the values for the **property_class** and **network_model_component**s considered in an **ato_aspect**, at one particular value for the *abscissa_class* (specified in the associated **ato_case**).

Attributes (in addition to those of **scan**)**:**

- none

Assertions (in addition to those of **scan**)**:**

- the *abscissa_value* shall be specified

Examples:

EXAMPLE 51 – A **scan_of_sampled_values** could record the temperatures of the nodes of a thermal mathematical model, at the beginning of an eclipse simulation in an ESATAN temperature calculation **run**.

4.2.65 security_classification_level

A **security_classification_level** is a specification of a category of security for access to data.

Attributes:

- *name*: label string that specifies the level of security

Assertions:

- Six levels of security are distinguished. The allowed levels are, in order of increasing confidentiality:
  - `'unclassified'`
  - `'classified'`
  - `'proprietary'`
  - `'confidential'`
  - `'secret'`
  - `'top_secret'`

  NOTE 50 – These security classification levels were taken from [STEP-203].

4.2.66 submodel_usage

A **submodel_usage** is a specification of an occurrence of one **network_model** as a submodel of another **network_model**, which then is the supermodel. Through this composition relationship a hierarchical model/submodel tree can be created – a supermodel is the 'whole' and its submodels are the 'parts'. The **submodel_usage** is referenced in the *constituents* attribute of a **network_model**.

NOTE 51 – The purpose of such a model/submodel tree is to support modular breakdown for network models and good configuration control by enabling a single source definition for the submodel occurrences of similar or identical engineering objects.

Attributes:

- *id:* identifier string
- *name:* label string, which specifies the usage of – i.e. the role played by – the *submodel* in relation to the supermodel in the model/submodel tree
- *submodel*: reference to a **network_model**
- *source_node*: reference to a **network_node** identifying one node of submodel that defines the origin of the submodel when it is mapped in another occurrence
- *target_node*: reference to a **network_node** identifying the node where *source_node* is mapped when the submodel usage is created

Assertions:

- The model/submodel tree established by **submodel_usage**s shall form an acyclic graph – i.e. no circular references is allowed
- *source_node* shall be in the set of constituents of *submodel*
- A **submodel_usage** shall be referred to by exactly one **network_model**

Examples:

EXAMPLE 52 – Typical examples for *name* are 'Unit 12', 'Lower –X Unit', 'Primary Supply', 'Secondary Supply'.

EXAMPLE 53 – 'Battery-1' and 'Battery-2' can be the *name*s for **submodel_usages** referencing a **network_model** with the *name* 'Battery'.

4.2.67 tabular_function

A **tabular_function** is a specialization of **parameterized_function**, which specifies the function's values in tabular form.

Attributes (in addition to those of **parameterized_function**)**:**

- *parameter_values*: one, two or three lists of a list of one or more **scalar_value**s, specifying the parameter values for each of the *parameter_properties*
- *function_values*: list of **scalar_value**
- *interpolation_type*: a **tabular_interpolation_type**
- *interpolation_degree*: an OPTIONAL integer ≥ 1, specifying the desired interpolation degree for polynomial interpolation

Assertions (in addition to those of **parameterized_function**)**:**

- The number of *parameter_properties* of the **parameterized_function** shall be no more than 3, i.e. a function of at most 3 variables.
- The *parameter_values* in the list for each *parameter_properties* shall be specified in monotonic ascending order.
- The size of the *function_values* list shall be equal to the product of the listsizes of each of the lists in *parameter_values.*
- The ordering of the *function_values* list shall be such that the index for the highest number *parameter_properties* changes fastest.

EXAMPLE 54 – A **tabular_function** $T(a_i,b_j)$ depends on two *parameter_properties*: *a* with index i and *b* with index j. For *a* 3 values are specified and for *b* 5 values are specified.
Then *parameter_values* has the following format: { $\{a_1, a_2, a_3\}$, $\{b_1, b_2, b_3, b_4, b_5\}$ },
and *function_values* is specified as: $\{T(a_1,b_1), T(a_1,b_2), T(a_1,b_3), T(a_1,b_4), T(a_1,b_5),$
$T(a_2,b_1), T(a_2,b_2), T(a_2,b_3), T(a_2,b_4), T(a_2,b_5), T(a_3,b_1), T(a_3,b_2), T(a_3,b_3), T(a_3,b_4), T(a_3,b_5)\}$

4.2.68 tabular_interpolation_type

A **tabular_interpolation_type** is a character string denoting the interpolation type belonging to a **tabular_function**. It can be one item from {polynomial, linear_logarithmic}.

There are no attributes nor assertions.

4.2.69 unit

A **unit** is a physical quantity, with a value of one, which is used as a standard in terms of which other quantities are expressed.

NOTE 52 – The definition of unit is taken directly from [STEP-41].

4.2.70 unit_assignment

A **unit_assignment** is a specification of the unit in which the values for a **property_class_scalar** are expressed, in the context of a given **ato_case**.

Attributes:

- *of_case*: a reference to the **ato_case**

- *for_property_class*: reference to the **property_class_scalar** for which a **unit** is assigned
- *assigned_unit*: reference to the **unit** that is assigned to values of *for_property_class*

Assertions:

- The combination of *of_case* and *for_property_class* shall be unique within the exchange dataset – i.e. there shall one and only one **unit_assignment** for a given **property_class_scalar** within the context of an **ato_case**.

Examples:

EXAMPLE 55 – A **unit_assignment** can assign the **unit** 'kelvin' to the **property_class_scalar_quantitative** with the *name* 'temperature'.

4.3 Application assertions

This subclause specifies the application assertions for the NRF application protocol. Application assertions specify the relations between application objects, the cardinality of the relationships, and the rules required for the integrity and validity of the application objects and UoFs. The application assertions and their definitions are given below.

4.3.1 abscissa_value_of_scans_with_invariant_lifetime

- The *abscissa_value* of a **scan** referenced by an **ato_aspect** for which the *lifetype* is invariant shall not be specified

4.3.2 consistent_units_per_ato_case

- All **unit_assignment**s referencing the same **ato_case** (through *of_case*) and **property_class_scalar**s with the same *name* – but possibly with different *roles* – shall reference the same **unit**. As a consequence the values of all these **property_class_scalar**s shall be expressed in identical units.

4.3.3 model_component_sequence_requires_ato_aspect

- A **model_component_sequence** shall be refered to by at least one **ato_aspect**

4.3.4 network_model_requires_product_definition

- When a **network_model** is defined, also the **product_version** (and **product**) that it represents shall be specified in the exchange dataset. At least one **model_represents_product_relationship** shall be specified which relates the **network_model** which is the top level supermodel to a top level **product_definition** of the **product_version** of the **product** that the model is representing.

4.3.5 network_node_in_one_network_model

- A **network_node** belongs to one and only one **network_model**, i.e. it shall only be referenced by only one **network_model** in its *constituents.*

4.3.6 network_node_relationship_in_one_network_model

- A **network_node_relationship** belongs to one and only one **network_model**, i.e. it shall only be referenced by only one **network_model** in its *constituents*.

4.3.7 approvals_are_assigned

- Any **approval** shall be assigned to another object.

Clause 5 - Application Interpreted Model

5.1 Mapping Table

This clause contains the mapping table that shows how each UoF and application object of this Part (see Clause 4) maps to one or several AIM resource constructs (see subclause 5.2 “AIM EXPRESS short listing”).

The mapping is not provided in the tabular format required by [STEP-AP-Guide], but in the simpler serial format presented below. This was done to improve the efficiency of the mapping activity, while keeping exactly the same information. The mapping is formatted as follows:

For each application object:

ARM_OBJECT <application_object_name>
IS_MAPPED_TO
<aim_element> USED_FROM <resource_schema>
[WITH_RULES <rule_number>[,<rule_number>...]]
WITH_REFERENCE_PATH
<reference_path>

And then for each attribute of the application object:

ATTRIBUTE <application_object_attribute_name>
IS_MAPPED_TO
<aim_element> USED_FROM <resource_schema>
WITH_REFERENCE_PATH
<reference_path>

The different elements are explained as follows:

- The <application_object_name> is the name of an application element as it appears in the application object definition in subclause 4.2. Attribute names are listed after the application object to which they belong.
- The <aim_element> is the name of an AIM element as it appears in the AIM (subclause 5.2), or the term ‘IDENTICAL MAPPING’ or the term ‘PATH’. AIM entities are written in lower case. Attribute names are referred to as <entity_name>.<attribute_name>. The mapping of an application element may result in several related AIM elements. Each of these AIM elements will require a line of its own in the reference path section. The term ‘IDENTICAL MAPPING’ indicates that both application objects of an application assertion map to the same AIM element. The term ‘PATH’ indicates that the application assertion maps to the entire reference path.
- The USED_FROM <resource_schema> gives, for those AIM elements that are interpreted from the integrated resources, the number of the corresponding Part of ISO 10303. For those AIM elements that are incorporated from an application interpreted construct (AIC), this is the AIC identifier, as defined below. For those AIM elements that are created for the purpose of this AP, this is the acronym of the Space-domain Integrated Resources or this AP – i.e. “SIR” and “NRF” respectively.
- Where applicable, in the WITH_RULES <rule_number>[,<rule_number>...] contruct one or several numbers may be given which refer to rules that apply to the current AIM element. For rules that are derived from relationships between application objects, the same rule is referred to by the mapping entries of all the involved AIM elements. The expanded names of the rules are listed after all mapping clauses.
- The WITH_REFERENCE_PATH <reference_path> construct is used to describe fully the mapping of an ARM element. It may be necessary to specify a reference path through several related AIM elements. A single AIM element is documented on a single line with a symbol which defines its relationship to the AIM element on the succeeding row in the column. The reference path column documents the role of an AIM element relative to the AIM element in the line succeeding it. Two or

more such related AIM elements define the interpretation of the Integrated Resources that satisfies the requirement specified by the application object. The reference path construct may be omitted when the path is trivial.

For each AIM element that has been created for use within this Part, a reference path up to its supertype from an integrated resource is specified.

- For the expression of reference paths and the relationships between AIM elements, the following notational conventions apply:

–>	attribute references ENTITY or SELECT type given in the following row;
<–	attribute is ENTITY or SELECT type referenced by the attribute in the following row;
=>	entity is a SUPERTYPE of the entity given in the following row;
<=	entity is a SUBTYPE of the entity given in the following row;
=	attribute has as its type the specified SELECT, STRING or ENUMERATION type, possibly constrained to particular choices or values.
[]	multiple AIM elements or sections of the reference path are required to satisfy an information requirement;
()	multiple AIM elements or sections of the reference path are identified as alternatives within the mapping to satisfy an information requirement;
{}	enclosed section constrains the reference path to satisfy an information requirement;
[i]	attribute is an aggregation of which a single member is given in the following row;
[n]	attribute is an aggregation of which member n is given in the following row;

Notes:

NOTE 53 – Readers are reminded that the reference path gives only a summary of the entities to be instantiated to cover the requirement defined by an application object. It does not explicitate computations which may be needed to translate an attribute defined in the ARM into attributes of entities defined in the AIM.

5.1.1 Mapping Table - product_structure UoF

5.1.1.1 ARM_OBJECT product

IS_MAPPED_TO
product USED_FROM 41
WITH_RULES 22

5.1.1.2 ARM_OBJECT product_context

IS_MAPPED_TO
product_context USED_FROM 41
WITH_RULES 5

ATTRIBUTE name
IS_MAPPED_TO
application_context_element.name USED_FROM 41
WITH_REFERENCE_PATH
product_context<=application_context_element
application_context_element.name

ATTRIBUTE discipline_type
IS_MAPPED_TO
product_context.discipline_type USED_FROM 41

5.1.1.3 ARM_OBJECT product_definition

IS_MAPPED_TO

product_definition USED_FROM 41
WITH_RULES 27

5.1.1.4 ARM_OBJECT product_definition_context

IS_MAPPED_TO
product_definition_context USED_FROM 41
WITH_RULES 5

5.1.1.5 ARM_OBJECT product_next_assembly_usage_relationship

IS_MAPPED_TO
next_assembly_usage_occurence USED_FROM 44
WITH_RULES 28

ATTRIBUTE id
IS_MAPPED_TO
product_definition_relationship.id USED_FROM 41
WITH_REFERENCE_PATH
next_assembly_usage_occurence <= assembly_component_usage
assembly_component_usage <= product_definition_usage
product_definition_usage <= product_definition_relationship
product_definition_relationship.id

product_next_assembly_usage_relationship to product_definition (as assembly)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
next_assembly_usage_occurence <= assembly_component_usage
assembly_component_usage <= product_definition_usage
product_definition_usage <= product_definition_relationship
product_definition_relationship.relating_product_definition -> product_definition

product_next_assembly_usage_relationship to product_definition (as constituent)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
next_assembly_usage_occurence <= assembly_component_usage
assembly_component_usage <= product_definition_usage
product_definition_usage <= product_definition_relationship
product_definition_relationship.related_product_definition -> product_definition

5.1.1.6 ARM_OBJECT product_version

IS_MAPPED_TO
SIR_product_version USED_FROM SIR

ATTRIBUTE id
IS_MAPPED_TO
product_definition_formation.id USED_FROM 41
WITH_REFERENCE_PATH
SIR_product_version <= product_definition_formation
product_definition_formation.id

ATTRIBUTE description
IS_MAPPED_TO
product_definition_formation.description USED_FROM 41
WITH_REFERENCE_PATH

SIR_product_version <= product_definition_formation
product_definition_formation.description

product_version to product (as of_product)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_product_version <= product_definition_formation
product_definition_formation.of_product -> product

product_version to product_definition (as definitions)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_product_version <= product_definition_formation
product_definition_formation <- product_definition.formation

5.1.2 Mapping Table - network_model_representation UoF

5.1.2.1 ARM_OBJECT model_represents_product_relationship

IS_MAPPED_TO
NRF_model_product_relationship USED_FROM NRF
WITH_RULES 23
WITH_REFERENCE_PATH
NRF_model_product_relationship <= property_definition_representation

model_represents_product_relationship to network_model_constituent (as model_constituent)
#1 if the constituent is a network_node
#2 if the constituent is a network_node_relationship
#3 if the constituent is a submodel_usage
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
property_definition_representation.used_representation -> representation
representation.items[1] -> network_model_constituent
#1 (network_model_constituent = SIR_network_node)
#2 (network_model_constituent = SIR_network_node_relationship)
#3 (network_model_constituent = SIR_submodel_usage)

model_represents_product_relationship to product_definition (as represented_product)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
property_definition_representation.definition -> property_definition
{property_definition.name = 'product_description_for_ato'}
property_definition.definition = characterized_definition
characterized_definition = characterized_product_definition
characterized_product_definition = product_definition

5.1.2.2 ARM_OBJECT network_model

IS_MAPPED_TO
SIR_model USED_FROM SIR
WITH_RULES 23, 29

ATTRIBUTE id
IS_MAPPED_TO
SIR_model.id USED_FROM SIR

ATTRIBUTE version_id
IS_MAPPED_TO
SIR_model.version_id USED_FROM SIR

ATTRIBUTE name
IS_MAPPED_TO
representation.name USED_FROM 43
WITH_REFERENCE_PATH
SIR_model <= representation
representation.name

ATTRIBUTE description
IS_MAPPED_TO
SIR_model.description USED_FROM SIR

network_model to security_classification_level (as security_class)
IS_MAPPED_TO
PATH
WITH_RULES 4
WITH_REFERENCE_PATH
SIR_model = security_classified_item
security_classified_item <- NRF_security_assignment.items[i]
NRF_security_assignment <= security_classification_assignment
security_classification_assignment.assigned_security_classification -> security_classification
security_classification.security_level -> security_classification_level

network_model to network_model_constituent (as constituents)
#1 if the constituent is a network_node
#2 if the constituent is a network_node_relationship
#3 if the constituent is a submodel_usage
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_model <= representation
representation.items[i] -> network_model_constituent
#1 (network_model_constituent = SIR_network_node)
#2 (network_model_constituent = SIR_network_node_relationship)
#3 (network_model_constituent = SIR_submodel_usage)

5.1.2.3 ARM_OBJECT network_node

IS_MAPPED_TO
SIR_node USED_FROM SIR

ATTRIBUTE id
IS_MAPPED_TO
SIR_node.id USED_FROM SIR

ATTRIBUTE name
IS_MAPPED_TO
representation_item.name USED_FROM 43
WITH_REFERENCE_PATH

SIR_node <= representation_item
representation_item.name

network_node to network_node_class (as class)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_node.classes[i] -> type_qualifier

network_node to security_classification_level (as security_class)
IS_MAPPED_TO
PATH
WITH_RULES 4
WITH_REFERENCE_PATH
SIR_node = security_classified_item
security_classified_item <- NRF_security_assignment.items[i]
NRF_security_assignment <= security_classification_assignment
security_classification_assignment.assigned_security_classification -> security_classification
security_classification.security_level -> security_classification_level

5.1.2.4 ARM_OBJECT network_node_class

IS_MAPPED_TO
type_qualifier USED_FROM 45
WITH_RULES 19

ATTRIBUTE name
IS_MAPPED_TO
type_qualifier.name USED_FROM 45

5.1.2.5 ARM_OBJECT network_node_relationship

IS_MAPPED_TO
SIR_node_relationship USED_FROM SIR

ATTRIBUTE id
IS_MAPPED_TO
SIR_node_relationship.id USED_FROM 43

ATTRIBUTE name
IS_MAPPED_TO
representation_item.name USED_FROM 43
WITH_REFERENCE_PATH
SIR_node_relationship <= representation_item
representation_item.name

network_node_relationship to network_node_relationship_class (as class)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR _node_relationship.classes[i] -> type_qualifier

network_node_relationship to security_classification_level (as security_class)
IS_MAPPED_TO
PATH
WITH_RULES 4
WITH_REFERENCE_PATH
SIR_node_relationship = security_classified_item

security_classified_item <- NRF_security_assignment.items[i]
NRF_security_assignment <= security_classification_assignment
security_classification_assignment.assigned_security_classification -> security_classification
security_classification.security_level -> security_classification_level

network_node_relationship to network_node (as nodes)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_node_relationship.nodes[i] -> SIR_node_or_usage
SIR_node_or_usage = SIR_node

network_node_relationship to network_node_usage (as nodes)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_node_relationship.nodes[i] -> SIR_node_or_usage
SIR_node_or_usage = SIR_node_usage

5.1.2.6 ARM_OBJECT network_node_relationship_class

IS_MAPPED_TO
type_qualifier USED_FROM 45
WITH_RULES 19

ATTRIBUTE name
IS_MAPPED_TO
type_qualifier.name USED_FROM 45

5.1.2.7 ARM_OBJECT network_node_relationship_usage

IS_MAPPED_TO
SIR_node_relationship_usage USED_FROM SIR

network_node_relationship_usage to submodel_usage (as used_submodel)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_node_relationship_usage.used_submodel -> SIR_submodel_usage

network_node_relationship_usage to network_node_relationship (as relationship)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_node_relationship_usage.component -> SIR_node_relationship

5.1.2.8 ARM_OBJECT network_node_usage

IS_MAPPED_TO
SIR_node_usage USED_FROM SIR

network_node_usage to submodel_usage (as used_submodel)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_node_usage.used_submodel -> SIR_submodel_usage

network_node_usage to network_node (as node)
IS_MAPPED_TO

PATH
WITH_REFERENCE_PATH
SIR_node_usage.node -> SIR_node

5.1.2.9 ARM_OBJECT submodel_usage

IS_MAPPED_TO
SIR_submodel_usage USED_FROM SIR

ATTRIBUTE id
IS_MAPPED_TO
SIR_submodel_usage.id USED_FROM SIR

ATTRIBUTE name
IS_MAPPED_TO
representation_item.name USED_FROM 43
WITH_REFERENCE_PATH
SIR_submodel_usage <= mapped_item
mapped_item <= representation_item
representation_item.name

submodel_usage to network_model (as submodel)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_submodel_usage <= mapped_item
mapped_item.mapping_source -> representation_map
representation_map.mapped_representation -> representation
representation => SIR_model

submodel_usage to network_node (as source_node)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_submodel_usage <= mapped_item
mapped_item.mapping_source -> representation_map
representation_map.mapping_origin -> representation_item
representation_item => SIR_node

submodel_usage to network_node (as target_node)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_submodel_usage <= mapped_item
mapped_item.mapping_target -> representation_item
representation_item => SIR_node

5.1.3 Mapping Table - bulk_results UoF

5.1.3.1 ARM_OBJECT ato_aspect

IS_MAPPED_TO
SIR_aspect USED_FROM SIR

ATTRIBUTE name
IS_MAPPED_TO
SIR_aspect.name USED_FROM SIR

ato_aspect to property_class_usage (as for_property_class)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_aspect.valued_property -> SIR_property_usage

ATTRIBUTE lifetime
IS_MAPPED_TO
SIR_aspect.lifetime USED_FROM SIR

ato_aspect to model_component_sequence (as component_sequence)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_aspect.component_sequence -> SIR_component_sequence

ato_aspect to scan (as scans)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_aspect.scans[i] -> SIR_scan

ato_aspect to security_classification_level (as security_class)
IS_MAPPED_TO
PATH
WITH_RULES 4
WITH_REFERENCE_PATH
SIR_aspect = security_classified_item
security_classified_item <- NRF_security_assignment.items[i]
NRF_security_assignment <= security_classification_assignment
security_classification_assignment.assigned_security_classification -> security_classification
security_classification.security_level -> security_classification_level

5.1.3.2 ARM_OBJECT ato_campaign

IS_MAPPED_TO
SIR_ato_campaign USED_FROM SIR

ato_campaign to organizational_project (as of_project)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_ato_campaign.of_project -> organizational_project

ATTRIBUTE name
IS_MAPPED_TO
group.name USED_FROM 41
WITH_REFERENCE_PATH
SIR_ato_campaign <= group
group.name

ATTRIBUTE description
IS_MAPPED_TO
group.description USED_FROM 41
WITH_REFERENCE_PATH
SIR_ato_campaign <= group

group.description

ato_campaign to date_and_time (as creation_date_and_time)
IS_MAPPED_TO
PATH
WITH_RULES 1
WITH_REFERENCE_PATH
SIR_ato_campaign = dated_item
dated_item<- NRF_date_assignment.dated_items[i]
NRF_date_assignment <= date_and_time_assignment
date_and_time_assignment.assigned_date -> date_and_time
{date_and_time_assignment.role -> date_time_role
date_time_role.name = 'CREATION_DATE'}

ato_campaign to date_and_time (as last_modification_date_and_time)
IS_MAPPED_TO
PATH
WITH_RULES 3
WITH_REFERENCE_PATH
SIR_ato_campaign = dated_item
dated_item<- NRF_date_assignment.dated_items[i]
NRF_date_assignment <= date_and_time_assignment
date_and_time_assignment.assigned_date -> date_and_time
{date_and_time_assignment.role -> date_time_role
date_time_role.name = 'LAST_MODIFICATION_DATE'}

ato_campaign to person_and_organization (as contact_person_organisation)
IS_MAPPED_TO
PATH
WITH_RULES 2, 17
WITH_REFERENCE_PATH
SIR_ato_campaign = sourced_item
sourced_item <- NRF_contact_assignment.items[i]
NRF_contact_assignment <= person_and_organization_assignment
person_and_organization_assignment.assigned_person_and_organization -> person_and_organization
person_and_organization_assignment.role -> person_and_organization_role
{person_and_organization_role.name = 'CONTACT_PERSON/ORGANIZATION'}

ato_campaign to approval (as approvals)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_ato_campaign = approved_item
approved_item <- NRF_approval_assignment.approved_items[i]
NRF_approval_assignment.assigned_approval -> approval
approval.status -> approval_status

ato_campaign to security_classification_level (as security_class)
IS_MAPPED_TO
PATH
WITH_RULES 4
WITH_REFERENCE_PATH
SIR_ato_campaign = security_classified_item
security_classified_item <- NRF_security_assignment.items[i]

NRF_security_assignment <= security_classification_assignment
security_classification_assignment.assigned_security_classification -> security_classification
security_classification.security_level -> security_classification_level

ATTRIBUTE application_interpreted_model_schema_name
IS_MAPPED_TO
application_protocol_definition.application_interpreted_model_schema_name USED_FROM 41
WITH_RULES 5
WITH_REFERENCE_PATH
SIR_ato_campaign.in_context -> application_context_element
application_context_element.frame_of_reference -> application_context
application_context <- application_protocol.application

ATTRIBUTE application_protocol_year
IS_MAPPED_TO
application_protocol_definition.application_protocol_year USED_FROM 41
WITH_REFERENCE_PATH
SIR_ato_campaign.in_context -> application_context_element
application_context_element.frame_of_reference -> application_context
application_context <- application_protocol.application
application_protocol_definition.application_protocol_year

5.1.3.3 ARM_OBJECT ato_case

IS_MAPPED_TO
SIR_ato_case USED_FROM SIR

ato_case to ato_campaign (as of_campaign)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_ato_case.campaign_membership -> SIR_case_in_campaign
SIR_case_in_campaign <= group_assignment
group_assignment.assigned_group -> group
group => SIR_ato_campaign

ATTRIBUTE id
IS_MAPPED_TO
SIR_ato_case.id USED_FROM SIR

ATTRIBUTE name
IS_MAPPED_TO
SIR_ato_case.name USED_FROM SIR

ATTRIBUTE description
IS_MAPPED_TO
SIR_ato_case.description USED_FROM SIR

ato_case to network_model (as root)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_ato_case.root_model -> SIR_model

ato_case to property_class_usage (as abscissa_class)
IS_MAPPED_TO
PATH

WITH_REFERENCE_PATH
SIR_ato_case.abscissa_class -> SIR_property_usage

ATTRIBUTE abscissa_sequencing
IS_MAPPED_TO
SIR_ato_case.abscissa_sequencing USED_FROM SIR

ato_case to ato_aspect (as predefined_aspects)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_ato_case = SIR_case_or_phase
SIR_case_or_phase <- SIR_initial_conditioning.conditioned_data
SIR_initial_conditioning.initial_condition -> SIR_aspect

ato_case to security_classification_level (as security_class)
IS_MAPPED_TO
PATH
WITH_RULES 4
WITH_REFERENCE_PATH
SIR_ato_case = security_classified_item
security_classified_item <- NRF_security_assignment.items[i]
NRF_security_assignment <= security_classification_assignment
security_classification_assignment.assigned_security_classification -> security_classification
security_classification.security_level -> security_classification_level

5.1.3.4 ARM_OBJECT ato_phase

IS_MAPPED_TO
SIR_ato_phase USED_FROM SIR

ATTRIBUTE id
IS_MAPPED_TO
SIR_ato_phase.id USED_FROM SIR

ATTRIBUTE name
IS_MAPPED_TO
SIR_ato_phase.name USED_FROM SIR

ATTRIBUTE description
IS_MAPPED_TO
SIR_ato_phase.description USED_FROM SIR

ato_phase to ato_case (as of_case)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_ato_phase.of_case -> SIR_ato_case

ato_phase to ato_aspect (as predefined_aspects)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_ato_phase = SIR_case_or_phase
SIR_case_or_phase <- SIR_initial_conditioning.conditioned_data
SIR_initial_conditioning.initial_condition -> SIR_aspect

ato_phase to security_classification_level (as security_class)
IS_MAPPED_TO
PATH
WITH_RULES 4
WITH_REFERENCE_PATH
SIR_ato_phase = security_classified_item
security_classified_item <- NRF_security_assignment.items[i]
NRF_security_assignment <= security_classification_assignment
security_classification_assignment.assigned_security_classification -> security_classification
security_classification.security_level -> security_classification_level

5.1.3.5 ARM_OBJECT list_of_descriptive_values

IS_MAPPED_TO
SIR_list_of_descriptive_values USED_FROM SIR

list_of_descriptive_values to property_class_usage (as applicable_for)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_list_of_ descriptive_values.applicable_for[i] -> SIR_applicability_of_values
SIR_applicability_of_values.valid_usage -> SIR_property_usage

ATTRIBUTE name
IS_MAPPED_TO
SIR_list_of_descriptive_values.name USED_FROM SIR

ATTRIBUTE entries
IS_MAPPED_TO
SIR_list_of_descriptive_values.entries USED_FROM SIR

5.1.3.6 ARM_OBJECT list_of_functions

IS_MAPPED_TO
SIR_list_of_functions USED_FROM SIR

list_of_functions to property_class_usage (as applicable_for)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_list_of_ functions.applicable_for[i] -> SIR_applicability_of_values
SIR_applicability_of_values.valid_usage -> SIR_property_usage

ATTRIBUTE name
IS_MAPPED_TO
SIR_list_of_functions.name USED_FROM SIR

list_of_functions to parameterized_functions (as entries)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_list_of_functions.entries[i] -> SIR_parameterized_function

5.1.3.7 ARM_OBJECT model_component_sequence

IS_MAPPED_TO
SIR_component_sequence USED_FROM SIR

ATTRIBUTE id
IS_MAPPED_TO
SIR_component_sequence.id USED_FROM SIR

ATTRIBUTE name
IS_MAPPED_TO
SIR_component_sequence.name USED_FROM SIR
WITH_RULES 11

model_component_sequence to network_model_component (as components)
IS_MAPPED_TO
#1 if the constituent is a network_node
#2 if the constituent is a network_node_relationship
#3 if the constituent is a submodel_usage
#4 if the constituent is a network_node_usage
#5 if the constituent is a network_node_relationship_usage
#6 if the constituent is a network_model
PATH
WITH_REFERENCE_PATH
NRF_component_sequence.components[i] -> SIR_network_component
#1 (SIR_network_component = SIR_node)
#2 (SIR_network_component = SIR_node_relationship)
#3 (SIR_network_component = SIR_submodel_usage)
#4 (SIR_network_component = SIR_node_usage)
#5 (SIR_network_component = SIR_node_relationship_usage)
#6 (SIR_network_component = SIR_model)

5.1.3.8 ARM_OBJECT property_class

IS_MAPPED_TO
SIR_property_class USED_FROM SIR

5.1.3.9 ARM_OBJECT property_class_scalar

IS_MAPPED_TO
SIR_property_scalar USED_FROM SIR

property_class_scalar to property_name (as name)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_property_scalar <= SIR_property_class
SIR_property_class.name -> SIR_property_name

ATTRIBUTE symmetry
IS_MAPPED_TO
SIR_property_scalar.symmetry USED_FROM SIR

5.1.3.10 ARM_OBJECT property_class_scalar_descriptive

IS_MAPPED_TO
SIR_property_descriptive USED_FROM SIR

5.1.3.11 ARM_OBJECT property_class_scalar_functional

IS_MAPPED_TO
SIR_property_functional USED_FROM SIR

5.1.3.12 ARM_OBJECT property_class_scalar_quantitative

IS_MAPPED_TO
SIR_property_quantitative USED_FROM SIR

5.1.3.13 ARM_OBJECT property_class_tensor

IS_MAPPED_TO
SIR_property_tensor USED_FROM SIR

property_class_tensor to property_name (as name)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_property_tensor <= SIR_property_class
SIR_property_class.name -> SIR_property_name

ATTRIBUTE tensor_order
IS_MAPPED_TO
SIR_property_tensor.order USED_FROM SIR

ATTRIBUTE dimensionality
IS_MAPPED_TO
SIR_property_tensor.dimensionality USED_FROM SIR

5.1.3.14 ARM_OBJECT property_class_usage

IS_MAPPED_TO
SIR_property_usage USED_FROM SIR

property_class_usage to ato_case (as of_case)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_property_usage.of_case -> SIR_ato_case

property_class_usage to property_class (as property)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_property_usage.property -> SIR_property_class

property_class_usage to property_role (as meaning)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_property_usage.meaning -> SIR_property_meaning
SIR_property_meaning.roles[i] -> type_qualifier

5.1.3.15 ARM_OBJECT property_name

IS_MAPPED_TO
SIR_property_name USED_FROM SIR

ATTRIBUTE name
IS_MAPPED_TO
SIR_property_name.name USED_FROM SIR

5.1.3.16 ARM_OBJECT property_role

IS_MAPPED_TO
type_qualifier USED_FROM 45

ATTRIBUTE name
IS_MAPPED_TO
type_qualifier.name USED_FROM 45

5.1.3.17 ARM_OBJECT run

IS_MAPPED_TO
NRF_run USED_FROM NRF
WITH_RULES 24
WITH_REFERENCE_PATH
NRF_run <= executed_action

run to ato_phase (as of_phase)
IS_MAPPED_TO
PATH
WITH_RULES 20
WITH_REFERENCE_PATH
NRF_run <= executed_action
executed_action <= action
action <- action_assignment.assigned_action
action_assignment => NRF_run_phase
NRF_run_phase.phase[1] -> SIR_ato_phase

ATTRIBUTE id
IS_MAPPED_TO
action.name USED_FROM 41
WITH_REFERENCE_PATH
executed_action <= action
action.name

ATTRIBUTE name
IS_MAPPED_TO
name_assignment.assigned_name USED_FROM 41
WITH_REFERENCE_PATH
NRF_run = NRF_named_item
NRF_named_item <- NRF_name_assignment.named_items[i]
NRF_name_assignment <= name_assignment
name_assignment.assigned_name

ATTRIBUTE description
IS_MAPPED_TO
action.description USED_FROM 41
WITH_REFERENCE_PATH
executed_action <= action
action.description

run to date_and_time (as timestamp)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
NRF_ato_run = dated_item

dated_item<- NRF_date_assignment.dated_items[i]
NRF_date_assignment <= date_and_time_assignment
date_and_time_assignment.assigned_date -> date_and_time
date_and_time_assignment.role -> date_time_role
{date_time_role.name = 'START_DATE_AND_TIME'}

ATTRIBUTE creator_tool_or_facility
IS_MAPPED_TO
action_resource.name USED_FROM 41
WITH_REFERENCE_PATH
executed_action <= action
action = supported_item
supported_item <- action_resource.usage[1]
{action_resource.kind -> action_resource_type
action_resource_type = 'TOOL_OR_FACILITY'}
action_resource.name

run to ato_aspect (as aspects)
IS_MAPPED_TO
PATH
WITH_RULES 20
WITH_REFERENCE_PATH
NRF_run <= executed_action
executed_action <= action
action <- action_assignment.assigned_action
action_assignment => NRF_result_assignment
NRF_result_assignment.results[i] -> SIR_aspect

5.1.3.18 ARM_OBJECT run_sequence

IS_MAPPED_TO
NRF_run_sequence USED_FROM NRF
WITH_RULES 24
WITH_REFERENCE_PATH
NRF_run_sequence <= action

run_sequence to ato_case (as of_case)
IS_MAPPED_TO
PATH
WITH_RULES 20
WITH_REFERENCE_PATH
NRF_run_sequence <= action
action <- action_assignment.assigned_action
action_assignment => NRF_run_case
NRF_run_case.case[1] -> SIR_ato_case

ATTRIBUTE id
IS_MAPPED_TO
action.name USED_FROM 41

ATTRIBUTE name
IS_MAPPED_TO
name_assignment.assigned_name USED_FROM 41
WITH_REFERENCE_PATH
NRF_run_sequence = NRF_named_item
NRF_named_item <- NRF_name_assignment.named_items[i]

NRF_name_assignment <= name_assignment
name_assignment.assigned_name

ATTRIBUTE description
IS_MAPPED_TO
action.description USED_FROM 41
WITH_REFERENCE_PATH
NRF_run_sequence <= action
action.description

run_sequence to run (as runs)
IS_MAPPED_TO
PATH
WITH_RULES 25
WITH_REFERENCE_PATH
NRF_run_sequence <= action
action <- action_relationship.relating_action
action_relationship.related_action -> action
action => executed_action
executed_action => NRF_run

5.1.3.19 ARM_OBJECT scalar_in_tensor

IS_MAPPED_TO
SIR_scalar_in_tensor USED_FROM SIR

scalar_in_tensor to property_class_tensor (as tensor)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_scalar_in_tensor.tensor -> SIR_property_tensor

scalar_in_tensor to property_class_scalar (as scalar)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_scalar_in_tensor.scalar -> SIR_property_scalar

ATTRIBUTE roles
IS_MAPPED_TO
type_qualifier USED_FROM 45
WITH_REFERENCE_PATH
SIR_scalar_in_tensor.roles -> SIR_property_meaning
SIR_property_meaning.roles[i] -> type_qualifier

5.1.3.20 ARM_OBJECT scan

IS_MAPPED_TO
SIR_scan USED_FROM NRF
WITH_RULES 21

scan to ato_aspect (as of_aspect)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_scan <- SIR_aspect.scans[i]

ATTRIBUTE abscissa_value
IS_MAPPED_TO
SIR_scan.abscissa_value USED_FROM NRF

scan to scan_mask (as mask)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_scan.mask -> SIR_scan_mask

ATTRIBUTE values
IS_MAPPED_TO
SIR_scan.values USED_FROM SIR

5.1.3.21 ARM_OBJECT scan_derivation_procedure

IS_MAPPED_TO
NRF_scan_derivation_procedure USED_FROM NRF
WITH_REFERENCE_PATH
NRF_scan_derivation_procedure <= action_method

ATTRIBUTE name
IS_MAPPED_TO
action_method.name USED_FROM 41

5.1.3.22 ARM_OBJECT scan_mask

IS_MAPPED_TO
SIR_mask USED_FROM NRF

ATTRIBUTE defined_ranges
IS_MAPPED_TO
SIR_mask.defined_ranges USED_FROM NRF

5.1.3.23 ARM_OBJECT scan_of_derived_values

IS_MAPPED_TO
NRF_derived_scan USED_FROM NRF
WITH_REFERENCE_PATH
NRF_derived_scan <= SIR_scan

scan_of_derived_values to scan_derivation_procedure (as derivation_procedure)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
NRF_derived_scan <- NRF_derivation.results[i]
NRF_derivation <= action_assignment
action_assignment.assigned_action -> action
action.chosen_method -> action_method
action_method => NRF_scan_derivation_procedure

ATTRIBUTE abscissa_start
IS_MAPPED_TO
NRF_derived_scan.abscissa_start USED_FROM NRF

ATTRIBUTE abscissa_end
IS_MAPPED_TO
NRF_derived_scan.abscissa_end USED_FROM NRF

5.1.3.24 ARM_OBJECT scan_of_sampled_values

IS_MAPPED_TO
NRF_scan_sampled USED_FROM NRF
WITH_REFERENCE_PATH
NRF_scan_sampled <= SIR_scan

5.1.3.25 ARM_OBJECT unit_assignment

IS_MAPPED_TO
SIR_unit_assignment USED_FROM SIR

unit_assignment to ato_case (as of_case)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_unit_assignment.in_case -> SIR_ato_case

unit_assignment to property_class_scalar (as for_property_class)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_unit_assignment.for_property -> SIR_property_scalar

unit_assignment to unit (as assigned_unit)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_unit_assignment.assigned_unit -> unit

5.1.4 Mapping Table - parameterized_function UoF

5.1.4.1 ARM_OBJECT cyclic_tabular_function

IS_MAPPED_TO
SIR_cyclic_tabular_function USED_FROM SIR

ATTRIBUTE period
IS_MAPPED_TO
SIR_cyclic_tabular_function.period USED_FROM SIR

5.1.4.2 ARM_OBJECT parameterized_function

IS_MAPPED_TO
SIR_parameterized_function USED_FROM SIR

ATTRIBUTE name
IS_MAPPED_TO
SIR_parameterized_function.name USED_FROM SIR

ATTRIBUTE description
IS_MAPPED_TO
SIR_parameterized_function.description USED_FROM SIR

parameterized_function to property_class_scalar (as parameter_properties)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_parameterized_function.parameters[i] -> SIR_variable

parameterized_function to property_class_scalar (as result_property)
IS_MAPPED_TO
PATH
WITH_REFERENCE_PATH
SIR_parameterized_function.result -> SIR_variable

5.1.4.3 ARM_OBJECT parameterized_function_with_formula

IS_MAPPED_TO
SIR_parameterized_function USED_FROM SIR

ATTRIBUTE formula
IS_MAPPED_TO
mathematical_string.text_representation USED_FROM 13584-42
WITH_REFERENCE_PATH
SIR_parameterized_function.formula -> mathematical_string
mathematical_string.text_representation

ATTRIBUTE language
IS_MAPPED_TO
NO MAPPING

ATTRIBUTE creator_tool_or_facility
IS_MAPPED_TO
NO MAPPING

5.1.4.4 ARM_OBJECT polynomial_function

IS_MAPPED_TO
SIR_polynomial_function USED_FROM SIR

ATTRIBUTE degree
IS_MAPPED_TO
int_literal USED_FROM 13584-20

ATTRIBUTE coefficients
IS_MAPPED_TO
real_litteral USED_FROM 13584-20

ATTRIBUTE lower_domain_bounds
IS_MAPPED_TO
[comparison_less_equal] USED_FROM 13584-20
[literal_number] USED_FROM 13584-20
WITH_REFERENCE_PATH
SIR_polynomial_function.lower_bound_expressions[i] -> less_or_greater
less_or_greater = comparison_less_equal
comparison_less_equal <= comparison_expression
comparison_expression <= binary_gen_expression
binary_gen_expression.operands[1] -> generic_expression
generic_expression => generic_literal
generic_literal => literal_number

ATTRIBUTE upper_domain_bounds
IS_MAPPED_TO
[comparison_greater_equal] USED_FROM 13584-20
[literal_number] USED_FROM 13584-20
WITH_REFERENCE_PATH

SIR_polynomial_function. upper_bounds_expressions[i] -> less_or_greater
less_or_greater = comparison_greater_equal
comparison_greater_equal <= comparison_expression
comparison_expression <= binary_gen_expression
binary_gen_expression.operands[1] -> generic_expression
generic_expression => generic_literal
generic_literal => literal_number

5.1.4.5 ARM_OBJECT tabular_function

IS_MAPPED_TO
SIR_tabular_function USED_FROM SIR

ATTRIBUTE parameter_values
IS_MAPPED_TO
list_ascending_real_literals USED_FROM SIR
WITH_REFERENCE_PATH
SIR_tabular_function.parameter_values[i] -> list_ascending_real_literals

ATTRIBUTE function_values
IS_MAPPED_TO
list_of_real_literals USED_FROM SIR
WITH_REFERENCE_PATH
SIR_tabular_function.result_values -> SIR_list_of_real_literal

ATTRIBUTE interpolation_type
IS_MAPPED_TO
SIR_tabular_function.interpolation_type USED_FROM SIR

ATTRIBUTE interpolation_degree
IS_MAPPED_TO
SIR_tabular_function.interpolation_degree USED_FROM SIR

5.1.5 Mapping Table - date_and_time UoF

5.1.5.1 ARM_OBJECT calendar_date

IS_MAPPED_TO
calendar_date USED_FROM 41

5.1.5.2 ARM_OBJECT coordinated_universal_time_offset

IS_MAPPED_TO
coordinated_universal_time_offset USED_FROM 41

5.1.5.3 ARM_OBJECT date

IS_MAPPED_TO
date USED_FROM 41
WITH_RULES 26, 14

5.1.5.4 ARM_OBJECT date_and_time

IS_MAPPED_TO
date_and_time USED_FROM 41
WITH_RULES 7, 15

5.1.5.5 ARM_OBJECT local_time

IS_MAPPED_TO
local_time USED_FROM 41

WITH_RULES mandatory_minute_and_second

5.1.6 Mapping Table - general_support UoF

5.1.6.1 ARM_OBJECT address

IS_MAPPED_TO
address USED_FROM 41

5.1.6.2 ARM_OBJECT approval

IS_MAPPED_TO
approval_status USED_FROM 41
WITH_RULES 6, 8, 10, 12

ATTRIBUTE level
IS_MAPPED_TO
approval_status.name USED_FROM 41

approval to date_and_time (as date_time)
IS_MAPPED_TO
PATH
WITH_RULES 7
WITH_REFERENCE_PATH
aproval_status <- approval.status
approval <- approval_date_time.dated_approval
approval_date_time.date_time -> date_time_select
date_time_select = date_and_time

approval to person_and_organization (as by_person_organization)
IS_MAPPED_TO
PATH
WITH_RULES 9
WITH_REFERENCE_PATH
aproval_status <- approval.status
approval <- approval_person_organization.authorized_approval
approval_person_organization.person_organization -> person_organization_select
person_organization_select = person_and_organization

5.1.6.3 ARM_OBJECT document

IS_MAPPED_TO
document USED_FROM 41

5.1.6.4 ARM_OBJECT document_type

IS_MAPPED_TO
document_type USED_FROM 41

5.1.6.5 ARM_OBJECT document_usage_constraint

IS_MAPPED_TO
document_usage_constraint USED_FROM 41

5.1.6.6 ARM_OBJECT organization

IS_MAPPED_TO
organization USED_FROM 41

5.1.6.7 ARM_OBJECT organization_address

IS_MAPPED_TO
organizational_address USED_FROM 41

5.1.6.8 ARM_OBJECT organizational_project

IS_MAPPED_TO
organizational_project USED_FROM 41

5.1.6.9 ARM_OBJECT person

IS_MAPPED_TO
person USED_FROM 41
WITH_RULES 16

5.1.6.10 ARM_OBJECT person_and_organization

IS_MAPPED_TO
person_and_organization USED_FROM 41

5.1.6.11 ARM_OBJECT personal_address

IS_MAPPED_TO
personal_address USED_FROM 41

5.1.6.12 ARM_OBJECT security_classification_level

IS_MAPPED_TO
WITH_RULES 4, 18
security_classification_level USED_FROM 41

5.2 AIM EXPRESS short listing

This clause specifies the EXPRESS schema that uses elements from the integrated resources and the AICs and contains the types, entity specializations, rules, and functions that are specific to this part. This clause also specifies modifications to the textual material for the constructs that are imported from the AICs and the integrated resources. The definitions and EXPRESS provided in the integrated resources for constructs used in the AIM may include select list items and subtypes which are not imported into the AIM. Requirements stated in the integrated resources which refer to such items and subtypes apply exclusively to those items which are imported into the AIM.

*)

```
SCHEMA NRF_schema;

USE FROM SIR_product_version_schema;

USE FROM SIR_parameterized_function_schema;

USE FROM SIR_network_model_schema;

USE FROM SIR_property_schema;

USE FROM SIR_ato_campaign_schema;

USE FROM application_context_schema(
      product_context,
      product_definition_context,
      application_protocol_definition
      );

USE FROM product_definition_schema(
      product_definition
      );

USE FROM product_structure_schema(
      next_assembly_usage_occurrence
      );

USE FROM product_property_definition_schema(
      property_definition
      );

USE FROM product_property_representation_schema(
      property_definition_representation
      );

USE FROM management_resources_schema(
      security_classification_assignment,
      name_assignment,
      date_and_time_assignment,
      approval_assignment,
      person_and_organization_assignment,
      action_assignment
      );

USE FROM document_schema(
      document,
      document_type,
      document_usage_constraint
      );

USE FROM security_classification_schema(
      security_classification_level
      );

USE FROM approval_schema(
      approval,
```

```
      approval_date_time,
      approval_person_organization,
      approval_status
      );

USE FROM action_schema(
      action,
      action_method,
      executed_action,
      action_resource,
      action_relationship
      );

USE FROM person_organization_schema(
      address,
      personal_address,
      organizational_address,
      organization,
      person,
      person_and_organization
      );

USE FROM date_time_schema(
      date_and_time,
      calendar_date,
      coordinated_universal_time_offset,
      local_time
      );

USE FROM measure_schema;

USE FROM ISO_13584_expressions_schema(
      int_literal,
      int_numeric_variable,
      real_numeric_variable,
      power_expression,
      plus_expression,
      mult_expression
      );

(*
```

NOTES:

1) The schemas referenced above can be found in the following Parts :

Schema name	*Part*
application_context_schema	ISO 10303-41
measure_schema	ISO 10303-41
product_property_definition_schema	ISO 10303-41
management_resources_schema	ISO 10303-41
product_property_representation_schema	ISO 10303-41
document_schema	ISO 10303-41
product_definition_schema	ISO 10303-41
person_organization_schema	ISO 10303-41
date_time_schema	ISO 10303-41
ISO_13584_expressions_schema	ISO 13584-20
SIR_product_version_schema	SIR
SIR_parameterized_function_schema	SIR
SIR_network_model_schema	SIR

Schema name	*Part*
SIR_property_schema	SIR
SIR_ato_campaign_schema	SIR

5.2.1 Fundamental concepts and choices

This subclause contains normative statements about the way to create valid instances of entities defined in the AIM schema. Other explanations and advices are provided in the informative annex K.

5.2.2 Parameters and result of functions

The parameters and the result of a parameterized function are considered in this protocol as variables.

The definition of a variable of a function has several aspects. A variable has a formal identification suitable for the expression of the function (e.g. y=F(x)). It has also a semantics associating it with a quantity.

The identification of a variable shall be mapped as a subtype of **numeric_variable** (defined in ISO 13584-20). The semantics shall be defined, in the context of this protocol, as a reference to a **SIR_property_usage**.

The combination of both aspects shall be transferred using **SIR_variable.**

5.2.3 Polynomial functions

The resources used to map the application object have a structure quite different from the structure proposed in the ARM.

A polynomial function shall be mapped as an instance of **SIR_polynomial_function**.

The polynom shall be transferred as an instance of **numeric_expression** where the components of the polynom shall be mapped using

- **plus_expression**, **mult_expression**, **power_expression** for the operations
- subtypes of **literal_number** for the coefficients,
- **SIR_variable** for the formal parameters,
- **comparison_less_equal** to describe the lower bounds,
- **comparison_greater_equal** to describe the upper bounds.

For the bounds, the first operand of the expression shall be a subtype of **literal_number**, the second shall be a **SIR_variable** defined as one of the parameters of the function.

The degree of the polynomial shall appear in a **power_expression** as an **int_literal**.

5.2.4 Properties

This protocol distinguishes the definition of a class of property and its usage in a given context.

For example, *Length* is a class of property; *Radius* is a usage of the class of property *Length* in a context where some axisymmetric geometry is described.

In this protocol, the context of usage of a property class shall be an ato_case.

Therefore, a class of property shall be transferred as a subtype of **SIR_property_class** and a usage of a property class shall be transferred as a **SIR_property_usage**.

5.2.5 NRF_schema type definitions

5.2.5.1 security_classified_item

The **security_classified_item** type provides a mechanism to refer to the entities that can have a security classification level associated with.

EXPRESS specification:

```
*)
TYPE security_classified_item = SELECT (
      SIR_model,
      SIR_node,
      SIR_node_relationship,
      SIR_aspect,
      SIR_ato_campaign,
```

```
      SIR_ato_case,
      SIR_ato_phase
      );
END_TYPE;
(*
```

5.2.5.2 dated_item

The **dated_item** type provides a mechanism to refer to the entities that can have a date associated with.

EXPRESS specification:

```
*)
TYPE dated_item = SELECT (
      SIR_ato_campaign,
      NRF_run
      );
END_TYPE;
(*
```

5.2.5.3 approved_item

The **approved_item** type provides a mechanism to refer to the entities that can have an approval associated with.

EXPRESS specification:

```
*)
TYPE approved_item = SELECT (
      SIR_ato_campaign
      );
END_TYPE;
(*
```

5.2.5.4 sourced_item

The **sourced_item** type provides a mechanism to refer to the entities that are associated with an information defining a contact person.

EXPRESS specification:

```
*)
TYPE sourced_item = SELECT (
      SIR_ato_campaign
      );
END_TYPE;
(*
```

5.2.5.5 named_item

The **named_item** type provides a mechanism to refer to the entities that can have a name associated with, with using **NRF_name_assignment**.

EXPRESS specification:

```
*)
TYPE named_item = SELECT (
      NRF_run,
      NRF_run_sequence
      );
END_TYPE;
(*
```

5.2.6 NRF_schema entity definitions

5.2.6.1 NRF_security_assignment

A **NRF_security_assignment** is a kind of **security_classification_assignment** that enables to assign a security_classification_level to **security_classified_item**s.

EXPRESS specification:

```
*)
ENTITY NRF_security_assignment
SUBTYPE OF (security_classification_assignment);
```

```
      items: SET[1:?] OF security_classified_item;
WHERE
      WR1:
SELF\security_classification_assignment.assigned_security_classification.security_
level.name IN
['unclassified','classified','proprietary','confidential','secret','top_secret'];
END_ENTITY;
(*
```

Attribute definitions:

items: specifies a set of **security_classified_item** to which a security level is assigned.

Formal proposition:

WR1: the name of the **security_level** assigned to items shall be one of the following values: 'unclassified','classified','proprietary','confidential','secret','top_secret'.

5.2.6.2 NRF_date_assignment

A **NRF_date_assignment** is a kind of **date_and_time_assignment** that enables to assign a **date_and_time** to **dated_item**s.

EXPRESS specification:

```
*)
ENTITY NRF_date_assignment
SUBTYPE OF (date_and_time_assignment);
      items: SET[1:?] OF dated_item;
END_ENTITY;
(*
```

Attribute definitions:

items: specifies a set of **dated_items** to which a **date_and_time** is assigned.

5.2.6.3 NRF_approval_assignment

A **NRF_approval_assignment** is a kind of **approval_assignment** that enables to assign an **approval** to **approved_item**s.

EXPRESS specification:

```
*)
ENTITY NRF_approval_assignment
SUBTYPE OF (approval_assignment);
      items: SET[1:?] OF approved_item;
END_ENTITY;
(*
```

Attribute definitions:

items: specifies a set of **approved_items** to which an **approval** is assigned.

5.2.6.4 NRF_contact_assignment

A **NRF_contact_assignment** is a kind of **person_and_organization_assignment** that enables to identify a contact for a **sourced_item**s.

EXPRESS specification:

```
*)
ENTITY NRF_contact_assignment
SUBTYPE OF (person_and_organization_assignment);
      items: SET[1:?] OF sourced_item;
WHERE
      WR1: SELF\person_and_organization_assignment.role.name='contact_person/
organization';
END_ENTITY;
(*
```

Attribute definitions:

items: specifies a set of **sourced_items** to which a contact **person_and_organization** is assigned.

Formal proposition:

WR1: the role of the assigned **person_and_organization** shall be 'contact_person/organization'.

5.2.6.5 NRF_name_assignment

A **NRF_name_assignment** is a kind of **name_assignment** that enables to assign a name to **named_item**s.

EXPRESS specification:

```
*)
ENTITY NRF_name_assignment
SUBTYPE OF (name_assignment);
      items: SET[1:?] OF named_item;
END_ENTITY;
(*
```

Attribute definitions:

items: specifies a set of **named_items** to which a name is assigned.

5.2.6.6 NRF_model_product_relationship

A **NRF_model_product_relationship** is a kind of property_definition_representation that enables to associate a **SIR_model** of of a constituent of a **SIR_model** with a **product_definition**, in order to specify that this representation or this element of representation represents a particular product.

EXPRESS specification:

```
*)
ENTITY NRF_model_product_relationship
SUBTYPE OF (property_definition_representation) ;
WHERE
      WR1:
SELF\property_definition_representation.definition.name='product_description_for_a
to';
      WR2: 'NRF_SCHEMA.PRODUCT_DEFINITION' IN
TYPEOF(SELF\property_definition_representation.definition.definition);
      WR3: ('NRF_SCHEMA.SIR_MODEL' IN
TYPEOF(SELF\property_definition_representation.used_representation)) XOR
((SIZEOF(SELF\property_definition_representation.used_representation.items)=1)
AND
(SIZEOF(['NRF_SCHEMA.SIR_NODE','NRF_SCHEMA.SIR_NODE_RELATIONSHIP','NRF_SCHEMA.SIR_
SUBMODEL_USAGE']*TYPEOF(SELF\property_definition_representation.used_representatio
n.items[1]))=1));
END_ENTITY;
(*
```

Formal propositions:

WR1: the name of the **property_definition** referred to by the **NRF_model_relationship** shall be 'product_description_for_ato';

WR2: the **property_definition** referred to by the **NRF_model_relationship** shall be a property of a **product_definition**;

WR3: the representation referred to by the **NRF_model_relationship** shall be a **SIR_model** or it shall contain only one element in items and the only element of items shall be of type **SIR_node** or **SIR_node_relationship** or **SIR_submodel_usage**.

5.2.6.7 NRF_run

A **NRF_run** is a kind of **executed_action** made with the data of a **SIR_ato_phase**.

EXPRESS specification:

```
*)
ENTITY NRF_run
SUBTYPE OF (executed_action);
DERIVE
      run_name:  SET OF label := get_names( SELF);
      creator_tool_or_facility: SET OF action_resource:= get_tools( SELF);
      timestamp: SET OF date_and_time := get_timestamps(SELF);
INVERSE
      subject_association: NRF_run_phase FOR assigned_action;
      results_association: NRF_run_results FOR assigned_action;
UNIQUE
      UR1: SELF\action.name, subject_association;
```

```
WHERE
      WR1: SIZEOF(run_name) = 1;
      WR2: SIZEOF(creator_tool_or_facility) = 1;
      WR3: SIZEOF(timestamp) = 1;
END_ENTITY;
(*
```

Attribute definitions:

run_name: specifies a set of labels assigned to the **NRF_run**;

creator_tool_or_facility: specifies a set of **action_resource** that refer to the **NRF_run**;

timestamp: specifies a set of **date_and_time** assigned to the **NRF_run**;

subject_association: specifies the **NRF_run_phase** that refers to the **NRF_run**;

results_association: specifies the **NRF_run_results** that refers to the **NRF_run**.

Formal propositions:

UR1: there shall not be more than one **NRF_run** with the same id and subject;

WR1: there shall be exactly one label assigned to the **NRF_run** as run_name;

WR2: there shall be at most one **action_resource** assigned to the **NRF_run**;

WR3: there shall be exactly one **date_and_time** assigned to the **NRF_run**.

5.2.6.8 NRF_run_phase

A **NRF_run_phase** is a kind of **action_assignment** that associates a **NRF_run** with the **SIR_phase** that is the subject of the run.

EXPRESS specification:

```
*)
ENTITY NRF_run_phase
SUBTYPE OF (action_assignment);
      subject: SIR_ato_phase;
      SELF\action_assignment.assigned_action: NRF_run;
END_ENTITY;
(*
```

Attribute definitions:

subject: specifies the **SIR_phase** for which the **NRF_run** has been executed.

SELF\action_assignment.assigned_action: specifies the considered **NRF_run**;

5.2.6.9 NRF_run_results

A **NRF_run_results** is a kind of **action_assignment** that associates a **NRF_run** with the **SIR_aspect** that results from the run.

EXPRESS specification:

```
*)
ENTITY NRF_run_results
SUBTYPE OF (action_assignment);
      results: SET[1:?] OF SIR_aspect;
      SELF\action_assignment.assigned_action: NRF_run;
END_ENTITY;
(*
```

Attribute definitions:

results: specifies the **SIR_aspects** that result from the **NRF_run**.

SELF\action_assignment.assigned_action: specifies the considered **NRF_run**;

Formal propositions:

WR1: the assigned_action shall be a **NRF_run**.

5.2.6.10 NRF_run_sequence

A **NRF_run_sequence** is a kind of **action** that collects a set of one or more **NRF_runs** dealing with **SIR_phases** belonging to the same **SIR_ato_case**.

EXPRESS specification:

```
*)
ENTITY NRF_run_sequence
```

```
SUBTYPE OF (action);
DERIVE
      sequence_name:  SET OF label := get_names( SELF);
INVERSE
      subject_association: NRF_run_sequence_case FOR assigned_action;
      sequence: SET[1:?] OF NRF_run_in_sequence FOR relating_action;
UNIQUE
      UR1: SELF\action.name, subject_association;
WHERE
      WR1: SIZEOF(sequence_name) = 1;
END_ENTITY;
(*
```

Attribute definitions:

sequence_name: specifies a set of labels assigned to the **NRF_run**;

subject_association: specifies the **NRF_run_sequence_case** that refers to the **NRF_run_sequence**;

sequence: specifies a set of one or more **NRF_run_in_sequence**s that specify the **NRF_runs** that are included in the **NRF_run_sequence**.

Formal propositions:

UR1: there shall not be more than one **NRF_run_sequence** with the same SELF\action.name and subject;

WR1: there shall be exactly one label assigned to the **NRF_run_sequence** as sequence_name;

5.2.6.11 NRF_run_sequence_case

A **NRF_run_sequence_case** is a kind of **action_assignment** that associates a **NRF_run_sequence** with the **SIR_ato_case** that is the subject of the runs included in the sequence.

EXPRESS specification:

```
*)
ENTITY NRF_run_sequence_case
SUBTYPE OF (action_assignment);
      subject: SIR_ato_case;
      SELF\action_assignment.assigned_action: NRF_run_sequence;
END_ENTITY;
(*
```

Attribute definitions:

subject: specifies the **SIR_ato_case** for which the **NRF_run_sequence** has been executed;

SELF\action_assignment.assigned_action: specifies the considered **NRF_run_sequence**;

5.2.6.12 NRF_run_in_sequence

A **NRF_run_in_sequence** is a kind of **action_relationship** that associates a **NRF_run** with the **NRF_run_sequence** that it is a member of.

EXPRESS specification:

```
*)
ENTITY NRF_run_in_sequence
SUBTYPE OF (action_relationship);
      SELF\action_relationship.relating_action: NRF_run_sequence;
      SELF\action_relationship.related_action: NRF_run;
WHERE
      WR1: related_action.subject_association.subject.of_case :=:
relating_action.subject_association.subject;
END_ENTITY;
(*
```

Attribute definitions:

SELF\action_relationship.relating_action: specifies the considered **NRF_run_sequence**;

SELF\action_relationship.related_action: specifies a **NRF_run** that is a member of the **NRF_run_sequence**.

Formal propositions:

WR1: the **SIR_ato_phase**, subject of the related **NRF_run** shall refer to the **SIR_ato_case** that is the subject of the **NRF_run_sequence**.

5.2.6.13 NRF_scan_sampled

A **NRF_scan_sampled** is a kind of **SIR_scan** where the values are recorded at one particular value of the abscissa.

EXPRESS specification:

```
*)
ENTITY NRF_scan_sampled
SUBTYPE OF (SIR_scan);
WHERE
      WR1: SELF\SIR_scan.aspect.lifetime <> invariant;
      WR2: EXISTS(SELF\SIR_scan.abscissa_value);
END_ENTITY;
(*
```

Formal proposition:

WR1: an instance of **NRF_scan_sampled** shall not exist if the lifetime of the associated **SIR_aspect** is invariant

WR2: the attribute abscissa_value shall be specified for any instance of **NRF_scan_sampled**.

5.2.6.14 NRF_scan_derived

A **NRF_scan_derived** is a kind of **SIR_scan** where the recorded values have been obtained from a derivation procedure.

EXPRESS specification:

```
*)
ENTITY NRF_scan_derived
SUBTYPE OF (SIR_scan);
INVERSE
      derivation: NRF_derivation_result FOR result;
END_ENTITY;
(*
```

Attribute definitions:

derivation: specifies the **NRF_derivation_result** that specifies the **NRF_derivation** that produced the **NRF_scan_derived**.

5.2.6.15 NRF_derivation_procedure

A **NRF_derivation_procedure** is a kind of **action_method** that defines a method that may be used to obtain **NRF_scan_derived**.

EXPRESS specification:

```
*)
ENTITY NRF_derivation_procedure
SUBTYPE OF (action_method);
UNIQUE
      UR1: SELF\action_method.name;
END_ENTITY;
(*
```

Formal proposition:

UR1: the name of the **NRF_derivation_procedure** shall be unique in the exchange dataset.

5.2.6.16 NRF_derivation_result

A **NRF_derivation_result** is a kind of **action_assignment** that associates the **NRF_scan_derived** with the **NRF_derivation** that produced it.

EXPRESS specification:

```
*)
ENTITY NRF_derivation_result
SUBTYPE OF (action_assignment);
      result: NRF_scan_derived;
      SELF\action_assignment.assigned_action: NRF_derivation;
END_ENTITY;
(*
```

Attribute definitions:

result: specifies the **NRF_scan_derived** that results from the derivation;

SELF\action_assignment.assigned_action: specifies the **NRF_derivation** that produced the **NRF_scan_derived**.

5.2.6.17 NRF_derivation_bounds

A **NRF_derivation_bounds** is a kind of **action_assignment** that associates the bounds of the abscissa used to get a **NRF_scan_derived** with the action of derivation.

EXPRESS specification:

```
*)
ENTITY NRF_derivation_bounds
SUBTYPE OF (action_assignment);
      bounds: SET[1:2] OF SIR_number_or_string;
      SELF\action_assignment.assigned_action: NRF_derivation;
END_ENTITY;
(*
```

Attribute definitions:

result: specifies the **NRF_scan_derived** that results from the derivation;

SELF\action_assignment.assigned_action : specifies the **NRF_derivation t**hat used the bounds for the derivation.

5.2.6.18 NRF_derivation

A **NRF_derivation** is a kind of **executed_action** that produced a **NRF_scan_derived**.

EXPRESS specification:

```
*)
ENTITY NRF_derivation
SUBTYPE OF (executed_action) ;
      SELF\action.chosen_method: NRF_derivation_procedure;
INVERSE
      bounds_association: NRF_derivation_bounds FOR assigned_action;
      result_association: NRF_derivation_result FOR assigned_action;
END_ENTITY;
(*
```

Attribute definitions:

SELF\action.chosen_method: specifies the **NRF_derivation_procedure** used to obtain the derived scan;

bounds_association: specifies the **NRF_derivation_bounds** that specifies the bounds considered for the derivation;

result_association: specifies the **NRF_derivation_result** that specifies the **NRF_scan_derived** that results form the derivation;

5.2.7 NRF_schema rule definitions

5.2.7.1 ato_campaign_requires_creation_date

The **ato_campaign_requires_creation_date** specifies that each instance of **SIR_ato_campaign** shall be referred to by one instance of **NRF_date_assignment** that relates it to its creation date.

EXPRESS specification:

```
*)
RULE ato_campaign_requires_creation_date FOR (SIR_ato_campaign,
NRF_date_assignment);
WHERE
      WR1: SIZEOF(QUERY(obj<* SIR_ato_campaign| SIZEOF(QUERY(nda <*
NRF_date_assignment| (obj IN nda.items) AND (nda.role.name = 'creation_date')))<>1
))= 0;
END_RULE;
(*
```

Argument definitions:

SIR_ato_campaign: specifies the set of all instances of **SIR_ato_campaign**;

NRF_date_assignment: specifies the set of all instances of **NRF_date_assignment**.

Formal proposition:

WR1: there shall not be any instance of **SIR_ato_campaign** that shall not be referred to by exactly one **NRF_date_assignment** having the role ‘creation_date’.

5.2.7.2 ato_campaign_requires_contact

The **ato_campaign_requires_creation_date** specifies that each instance of **SIR_ato_campaign** shall be referred to by one instance of **NRF_contact_assignment** that relates it to a contact person/ organization.

EXPRESS specification:

```
*)
RULE ato_campaign_requires_contact FOR (SIR_ato_campaign, NRF_contact_assignment);
WHERE
     WR1: SIZEOF(QUERY(obj<* SIR_ato_campaign| SIZEOF(QUERY(nca <*
NRF_contact_assignment | obj IN nca.items))<>1 ))= 0;
END_RULE;
(*
```

Argument definitions:

SIR_ato_campaign: specifies the set of all instances of **SIR_ato_campaign**;

NRF_contact_assignment: specifies the set of all instances of **NRF_contact_assignment**;

Formal proposition:

WR1: there shall not be any instance of **SIR_ato_campaign** that shall not be referred to by exactly one **NRF_contact_assignment**.

5.2.7.3 at_most_one_modification_date

The **ato_campaign_requires_creation_date** specifies that each instance of **SIR_ato_campaign** shall be referred to by at most one instance of **SIR_ato_campaign** that relates it to a modification date.

EXPRESS specification:

```
*)
RULE at_most_one_modification_date FOR (SIR_ato_campaign, NRF_date_assignment);
WHERE
     WR1: SIZEOF(QUERY(obj<* SIR_ato_campaign| SIZEOF(QUERY(nda <*
NRF_date_assignment| (obj IN nda.items) AND (nda.role.name =
‘last_modification_date’)))>1 ))= 0;
END_RULE;
(*
```

Argument definitions:

SIR_ato_campaign: specifies the set of all instances of **SIR_ato_campaign**;

NRF_date_assignment: specifies the set of all instances of **NRF_date_assignment**.

Formal proposition:

WR1: there shall not be any instance of **SIR_ato_campaign** that shall not be referred to more than one **NRF_date_assignment** having the role ‘last_modification_date’.

5.2.7.4 at_most_one_security_level

The rule **at_most_one_security_level** specifies that each instance of **SIR_model**, **SIR_node**, **SIR_node_relationship**, **SIR_ato_campaign**, **SIR_ato_case** and **SIR_ato_phase** shall be referred to by at most one instance of **NRF_security_assignment** that relates it to a security level.

EXPRESS specification:

```
*)
RULE at_most_one_security_level FOR (SIR_model, SIR_node, SIR_node_relationship,
SIR_aspect, SIR_ato_campaign, SIR_ato_case, SIR_ato_phase,
NRF_security_assignment);
WHERE
     WR1: SIZEOF(QUERY(obj<* SIR_model| SIZEOF(QUERY(nsa <*
NRF_security_assignment | obj IN nsa.items))>1 ))= 0;
     WR2: SIZEOF(QUERY(obj<* SIR_node| SIZEOF(QUERY(nsa <* NRF_security_assignment
| obj IN nsa.items))>1 ))= 0;
     WR3: SIZEOF(QUERY(obj<* SIR_node_relationship| SIZEOF(QUERY(nsa <*
NRF_security_assignment | obj IN nsa.items))>1 ))= 0;
```

```
     WR4: SIZEOF(QUERY(obj<* SIR_ato_campaign| SIZEOF(QUERY(nsa <*
NRF_security_assignment | obj IN nsa.items))>1 ))= 0;
     WR5: SIZEOF(QUERY(obj<* SIR_ato_case| SIZEOF(QUERY(nsa <*
NRF_security_assignment | obj IN nsa.items))>1 ))= 0;
     WR6: SIZEOF(QUERY(obj<* SIR_ato_phase| SIZEOF(QUERY(nsa <*
NRF_security_assignment | obj IN nsa.items))>1 ))= 0;
END_RULE;
(*
```

Argument definitions:

SIR_model: specifies the set of all instances of **SIR_model**.

SIR_node: specifies the set of all instances of **SIR_node**.

SIR_node_relationship: specifies the set of all instances of **SIR_node_relationship**.

SIR_ato_campaign: specifies the set of all instances of **SIR_ato_campaign**.

SIR_ato_case: specifies the set of all instances of **SIR_ato_case**.

SIR_ato_phase: specifies the set of all instances of **SIR_ato_phase**.

NRF_date_assignment: specifies the set of all instances of **NRF_date_assignment**.

Formal proposition:

WR1: there shall not be more than one security level assigned to a **SIR_model**;

WR2: there shall not be more than one security level assigned to a **SIR_node**;

WR3: there shall not be more than one security level assigned to a **SIR_node_relationship**;

WR4: there shall not be more than one security level assigned to a **SIR_ato_campaign**;

WR5: there shall not be more than one security level assigned to a **SIR_ato_case**;

WR6: there shall not be more than one security level assigned to a **SIR_ato_phase**;

5.2.7.5 application_context_requires_ap_definition

The rule **application_context_requires_ap_definition** specifies that any instance of **application_context** shall be referred to by exactly one instance of **application_protocol_definition**.

EXPRESS specification:

```
*)
RULE application_context_requires_ap_definition FOR
  (application_context, application_protocol_definition);
WHERE
  WR1: SIZEOF (QUERY (ac <* application_context |
       NOT (SIZEOF (QUERY (apd <* application_protocol_definition | ac :=:
apd.application)) = 1 ))) = 0;
END_RULE; -- application_context_requires_ap_definition
(*
```

Argument definitions:

application_context: specifies the set of all instances of **application_context**.

application_protocol_definition: specifies the set of all instances of **application_protocol_definition**.

Formal proposition:

WR1: there shall not be any instance of **application_context** that is not referred to by exactly one instance of **application_protocol_definition**.

5.2.7.6 approval_requires_approval_date_time

The rule **approval_requires_approval_date_time** specifies that each instance of **approval** shall be referred to by exactly one instance of **approval_date_time**.

EXPRESS specification:

```
*)
RULE approval_requires_approval_date_time FOR (approval,
         approval_date_time);
WHERE
  WR1: SIZEOF (QUERY ( app <* approval |
       NOT (SIZEOF (QUERY (adt <* approval_date_time |
       app :=: adt.dated_approval)) = 1))) = 0;
```

```
END_RULE; -- approval_requires_approval_date_time
(*
```

Argument definitions:

approval: specifies the set of all instances of **approval**.

approval_date_time: specifies the set of all instances of **approval_date_time**.

Formal proposition:

WR1: there shall not be any instance of **approval** that is not referred to by exactly one **approval_date_time**.

5.2.7.7 approval_date_time_requires_date_time

The rule **approval_date_time_requires_date_time** specifies that each instance of **approval_date_time** shall refer to an instance of date_and_time.

EXPRESS specification:

```
*)
RULE approval_date_time_requires_date_time FOR (approval_date_time);
WHERE
  WR1: SIZEOF (QUERY ( app <* approval_date_time |
       NOT ('NRF_SCHEMA.DATE_AND_TIME' IN TYPEOF(app.date_time)))) = 0;
END_RULE;
(*
```

Argument definitions:

approval_date_time: specifies the set of all instances of **approval_date_time**.

Formal proposition:

WR1: there shall not be any instance of **approval_date_time** that does not refer to a **date_and_time**.

5.2.7.8 approval_requires_approval_person_organization

The rule **approval_requires_approval_person_organization** specifies that each instance of **approval** shall be referred to by exactly one instance of **approval_person_organization**.

EXPRESS specification:

```
*)
RULE approval_requires_approval_person_organization FOR (approval,
        approval_person_organization);
WHERE
  WR1: SIZEOF (QUERY (app <* approval |
       NOT (SIZEOF (QUERY (apo <* approval_person_organization |
       app :=: apo.authorized_approval)) = 1))) = 0;
END_RULE; -- approval_requires_approval_person_organization
(*
```

Argument definitions:

approval: specifies the set of all instances of **approval**.

approval_person_organization: specifies the set of all instances of **approval_person_organization**.

Formal proposition:

WR1: there shall not be any instance of **approval** that is not referred to by exactly one **approval_person_organization**.

5.2.7.9 approval_person_organization_requires_person_organization

The rule **approval_person_organization_requires_person_organization** specifies that each instance of **approval_person_organization** shall refer to an instance of **person_and_organization**.

EXPRESS specification:

```
*)
RULE approval_person_organization_requires_person_organization FOR
(approval_person_organization);
WHERE
  WR1: SIZEOF (QUERY ( app <* approval_person_organization |
       NOT ('NRF_SCHEMA.PERSON_AND_ORGANIZATION' IN
TYPEOF(app.person_organization)))) = 0;
END_RULE;
```

```
(*
```

Argument definitions:

approval_person_organization: specifies the set of all instances of **approval_person_organization**.

Formal proposition:

WR1: there shall not be any instance of **approval_person_organization** that does not refer to a **person_and_organization**.

5.2.7.10 approvals_are_assigned

The rule **approvals_are_assigned** specifies that each instance of **approval** shall be referred to at least one instance of NRF_**approval_assignment** that relates it to the approved objects.

EXPRESS specification:

```
*)
RULE approvals_are_assigned FOR
  (approval, NRF_approval_assignment);
WHERE
  WR1: SIZEOF (QUERY (app <* approval |
       NOT (SIZEOF (QUERY (aa <* NRF_approval_assignment |
       app :=: aa.assigned_approval )) >= 1 ))) = 0;
END_RULE; -- approvals_are_assigned
(*
```

Argument definitions:

approval: specifies the set of all instances of **approval**.

NRF_approval_assignment: specifies the set of all instances of **NRF_approval_assignment**.

Formal proposition:

WR1: there shall not be any instance of **approval** that is not referred to by at least one **NRF_approval_assignment**.

5.2.7.11 component_sequence_requires_aspect

The rule **component_sequence_requires_aspect** specifies that each instance of **SIR_component_sequence** shall be referred to by at least one **SIR_aspect**.

EXPRESS specification:

```
*)
RULE component_sequence_requires_aspect FOR (SIR_component_sequence);
WHERE
      WR1: SIZEOF(USEDIN(SELF,  'NRF_SCHEMA.SIR_ASPECT.COMPONENT_SEQUENCE')) >0;
END_RULE;
(*
```

Argument definitions:

SIR_component_sequence: specifies the set of all instances of **SIR_component_sequence**.

Formal proposition:

WR1: any **SIR_component_sequence** shall be referred to by at least one **SIR_aspect**.

5.2.7.12 dependent_instantiable_approval_role

The rule **dependent_instantiable_approval_role** specifies that each instance of **approval_role** shall be referred to by at least another instance.

EXPRESS specification:

```
*)
RULE dependent_instantiable_approval_role FOR (approval_role);
WHERE
  WR1: SIZEOF (QUERY (dtr <* approval_role |
       NOT (SIZEOF (USEDIN (dtr, '')) >= 1))) = 0;
END_RULE;
(*
```

Argument definitions:

approval_role: specifies the set of all instances of **approval_role**.

Formal proposition:

WR1: there shall not be any instance of **approval_role** that is not referred to by at least one instance.

5.2.7.13 dependent_instantiable_approval_status

The rule **dependent_instantiable_approval_status** specifies that each instance of **approval_status** shall be referred to by at least another instance.

EXPRESS specification:

```
*)
RULE dependent_instantiable_approval_status FOR (approval_status);
WHERE
WR1: SIZEOF (QUERY (ast <* approval_status |
     NOT (SIZEOF (USEDIN (ast, '')) >= 1))) = 0;
END_RULE; -- dependent_instantiable_approval_status
(*
```

Argument definitions:

approval_status: specifies the set of all instances of **approval_status**.

Formal proposition:

WR1: any instance of **approval_statu**s shall be referred to by at least one instance.

5.2.7.14 dependent_instantiable_date

The rule **dependent_instantiable_date** specifies that each instance of **date** shall be referred to by at least an instance of **date_and_time**.

EXPRESS specification:

```
*)
RULE dependent_instantiable_date FOR (date);
WHERE
      WR1: SIZEOF (QUERY (dt <* date |
       NOT (SIZEOF(USEDIN (dt, 'NRF_SCHEMA.DATE_AND_TIME.DATE_COMPONENT')) >= 1)))
= 0;
END_RULE; -- dependent_instantiable_date
(*
```

Argument definitions:

date: specifies the set of all instances of **date**.

Formal proposition:

WR1: there shall not be any instance of **date** that is not referred to by at least one instance of **date_and_time**.

5.2.7.15 dependent_instantiable_date_time_role

The rule **dependent_instantiable_date_time_role** specifies that each instance of **date_time_role** shall be referred to by at least another instance.

EXPRESS specification:

```
*)
RULE dependent_instantiable_date_time_role FOR (date_time_role);
WHERE
  WR1: SIZEOF (QUERY (dtr <* date_time_role |
       NOT (SIZEOF (USEDIN (dtr, '')) >= 1))) = 0;
END_RULE; -- dependent_instantiable_date_time_role
(*
```

Argument definitions:

date_time_role: specifies the set of all instances of **date_time_role**.

Formal proposition:

WR1: there shall not be any instance of **date_time_role** that is not referred to by at least one instance.

5.2.7.16 dependent_instantiable_person

The rule **dependent_instantiable_person** specifies that each instance of **person** shall be referred to by at least an instance of **person_and_organization**.

EXPRESS specification:

```
*)
RULE dependent_instantiable_person FOR (person);
WHERE
      WR1: SIZEOF(USEDIN(SELF,  'NRF_SCHEMA.PERSON_AND_ORGANIZATION.THE_PERSON'))
>0;
END_RULE;
(*
```

Argument definition:

person: specifies the set of all instances of **person**.

Formal proposition:

WR1: any instance of **person** shall be referred to by at least one **person_and_organization**.

5.2.7.17 dependent_instantiable_person_and_organization_role

The rule **dependent_instantiable_person_and_organization_role** specifies that each instance of **person_and_organization_role** shall be referred to by at least another instance.

EXPRESS specification:

```
*)
RULE dependent_instantiable_person_and_organization_role FOR (
         person_and_organization_role);
WHERE
  WR1: SIZEOF (QUERY (poar <* person_and_organization_role |
       NOT (SIZEOF (USEDIN (poar, '')) >= 1))) = 0;
END_RULE; -- dependent_instantiable_person_and_organization_role
(*
```

Argument definitions:

person_and_organization_role: specifies the set of all instances of **person_and_organization_role**.

Formal proposition:

WR1: there shall not be any instance of **person_and_organization_role** that is not referred to by at least one instance.

5.2.7.18 dependent_instantiable_security_classification_level

The rule **dependent_instantiable_security_classification_level** specifies that each instance of **security_classification_level** shall be referred to by at least another instance.

EXPRESS specification:

```
*)
RULE dependent_instantiable_security_classification_level FOR
(security_classification_level);
WHERE
  WR1: SIZEOF (QUERY (scl <* security_classification_level |
       NOT (SIZEOF (USEDIN (scl, '')) >= 1))) = 0;
END_RULE; -- dependent_instantiable_security_classification_level
(*
```

Argument definitions:

security_classification_level: specifies the set of all instances of **security_classification_level**.

Formal proposition:

WR1: there shall not be any instance of **security_classification_level** that is not referred to by at least one instance.

5.2.7.19 dependent_instantiable_type_qualifier

The rule **dependent_instantiable_type_qualifier** specifies that each instance of **type_qualifier** shall be referred to by at least another instance.

EXPRESS specification:

```
*)
RULE dependent_instantiable_type_qualifier FOR (type_qualifier);
WHERE
  WR1: SIZEOF (QUERY (poar <* type_qualifier|
       NOT (SIZEOF (USEDIN (poar, '')) >= 1))) = 0;
END_RULE;
```

```
(*
```

Argument definitions:

type_qualifier: specifies the set of all instances of **type_qualifier**.

Formal proposition:

WR1: there shall not be any instance of **type_qualifier** that is not referred to by at least one instance.

5.2.7.20 exclusive_subtypes_action_assignment

The rule **exclusive_subtypes_action_assignment** specifies that any instance of **action_assignment** shall be either a **NRF_run_phase**, a **NRF_run_results**, a **NRF_run_sequence_case**, a **NRF_derivation_result** or a **NRF_derivation_bounds**.

EXPRESS specification:

```
*)
RULE exclusive_subtypes_action_assignment FOR (action_assignment);
WHERE
      WR1: SIZEOF (QUERY (act <* action_assignment |
SIZEOF(['NRF_SCHEMA.NRF_RUN_PHASE','NRF_SCHEMA.NRF_RUN_RESULTS',’NRF_SCHEMA.NRF_RU
N_SEQUENCE_CASE’, ‘NRF_SCHEMA.NRF_DERIVATION_RESULT’,
‘NRF_SCHEMA.NRF_DERIVATION_BOUNDS’]*TYPEOF(act))<>1)) = 0;
END_RULE;
(*
```

Argument definitions:

action_assignment: specifies the set of all instances of **action_assignment**.

Formal proposition:

WR1: there shall not be any instance of **action_assignment** that is not an instance either of **NRF_run_phase** or of **NRF_RUN_results** or of **NRF_run_sequence_case** or of **NRF_derivation_result** or of **NRF_derivation_bounds**.

5.2.7.21 exclusive_ subtypes_of_scan

The rule **exclusive_ subtypes_of_scan** specifies that an instance of **NRF_scan_derived** shall not be simultaneously an instance of **NRF_scan_sampled**.

EXPRESS specification:

```
*)
RULE exclusive_subtypes_of_scan FOR (SIR_scan) ;
WHERE
      WR1: SIZEOF(QUERY(ss <* SIR_scan |
 ((('NRF_SCHEMA.NRF_SCAN_DERIVED') IN TYPEOF(ss)) AND
(('NRF_SCHEMA.NRF_SCAN_SAMPLED') IN TYPEOF(ss))) ))  =0;
END_RULE;
(*
```

Argument definitions:

SIR_scan: specifies the set of all instances of **SIR_scan**.

Formal propositions:

WR1: any instance of **SIR_scan** shall not be simultaneously an instance of **SIR_scan_derived** and an instance of **SIR_scan_sampled**.

5.2.7.22 product_requires_version

The rule **product_requires_version** specifies that any instance of **product** shall be referred to by at least one **SIR_product_version**.

EXPRESS specification:

```
*)
RULE product_requires_version FOR (product, SIR_product_version);
WHERE
  WR1: SIZEOF (QUERY (prod <* product |
       NOT (SIZEOF (QUERY (pdf <* SIR_product_version |
       prod :=: pdf\product_definition_formation.of_product )) >= 1 ))) = 0;
END_RULE; -- product_requires_version
(*
```

Argument definitions:

product: specifies the set of all instances of **product**.

SIR_product_version: specifies the set of all instances of **SIR_product_version**.

Formal proposition:

WR1: there shall not be any instance of **product** that is not referred to by at least one **SIR_product_version**.

5.2.7.23 model_requires_definition

The rule **model_requires_definition** specifies that any instance of **SIR_model** shall be referred to by at least one **NRF_model_product_relationship**.

EXPRESS specification:

```
*)
RULE model_requires_definition FOR (SIR_model, NRF_model_product_relationship);
WHERE
      WR1: SIZEOF(QUERY(sm <* SIR_model | SIZEOF(QUERY(nmpr <*
NRF_model_product_relationship | sm :=:
nmpr\property_definition_representation.used_representation))=0))=0;
END_RULE;
(*
```

Argument definitions:

SIR_model: specifies the set of all instances of **SIR_model**.

NRF_model_product_relationship: specifies the set of all instances of **NRF_model_product_relationship**.

Formal proposition:

WR1: there shall not be any instance of **SIR_model** that is not referred to by at least one instance of **NRF_model_product_relationship**.

5.2.7.24 subtype_mandatory_action

The rule **subtype_mandatory_action** specifies that only instances of the subtypes of **action** are permitted.

EXPRESS specification:

```
*)
RULE subtype_mandatory_action FOR (action);
WHERE
      WR1: SIZEOF (QUERY (act <* action |
SIZEOF(['NRF_SCHEMA.NRF_RUN','NRF_SCHEMA.NRF_RUN_SEQUENCE',
'NRF_SCHEMA.NRF_DERIVATION']*TYPEOF(act))<>1)) = 0;
END_RULE; -- subtype_mandatory_action
(*
```

Argument definitions:

action: specifies the set of all instances of **action**.

Formal proposition:

WR1: there shall not be any instance of **action** that is not an instance either of **NRF_run** or **NRF_run_sequence** or of **NRF_derivation**.

5.2.7.25 subtype_mandatory_action_relationship

The rule **subtype_mandatory_action_relationship** specifies that only instances of the subtypes of **action_relationship** are permitted.

EXPRESS specification:

```
*)
RULE subtype_mandatory_action_relationship FOR (action_relationship);
WHERE
      WR1: SIZEOF (QUERY (act <* action_relationship |
NOT ('NRF_SCHEMA.NRF_RUN_IN_SEQUENCE' IN TYPEOF(act)))) = 0;
END_RULE;
(*
```

Argument definitions:

action_relationship: specifies the set of all instances of **action_relationship**.

Formal proposition:

WR1: there shall not be any instance of **action_relationship** that is not an instance of **NRF_run_in_sequence**.

5.2.7.26 subtype_mandatory_date

The rule **subtype_mandatory_date** specifies that only instances of the subtype **calendar_date** of **date** are permitted.

EXPRESS specification:

```
*)
RULE subtype_mandatory_date FOR (date);
WHERE
     WR1: SIZEOF (QUERY (act <* date|NOT('NRF_SCHEMA.CALENDAR_DATE' IN
TYPEOF(act)))) = 0;
END_RULE;
(*
```

Argument definitions:

date: specifies the set of all instances of **date**.

Formal proposition:

WR1: there shall not be any instance of **date** that is not an instance of **calendar_date**.

5.2.7.27 subtype_mandatory_product_definition_formation

The rule **subtype_mandatory_product_definition_formation** specifies that only instances of the subtype **SIR_product_version** of **product_definition_formation** are permitted.

EXPRESS specification:

```
*)
RULE subtype_mandatory_product_definition_formation FOR
(product_definition_formation);
WHERE
     WR1: SIZEOF(QUERY(obj<*
product_definition_formation|NOT('NRF_SCHEMA.SIR_PRODUCT_VERSION' IN
TYPEOF(obj))))= 0;
END_RULE;
(*
```

Argument definitions:

product_definition_formation: specifies the set of all instances of **product_definition_formation**.

Formal proposition:

WR1: there shall not be any instance of **product_definition_formation** that is not an instance of **SIR_product_version**.

5.2.7.28 subtype_mandatory_product_definition_relationship

The rule **subtype_mandatory_product_definition_relationship** specifies that only instances of the subtype **next_assembly_usage_occurrence** of **product_definition_relationship** are permitted.

EXPRESS specification:

```
*)
RULE subtype_mandatory_product_definition_relationship FOR
(product_definition_relationship);
WHERE
     WR1: SIZEOF(QUERY(obj<*
product_definition_relationship|NOT('NRF_SCHEMA.NEXT_ASSEMBLY_USAGE_OCCURENCE' IN
TYPEOF(obj))))= 0;
END_RULE;
(*
```

Argument definitions:

product_definition_relationship: specifies the set of all instances of **product_definition_relationship**.

Formal proposition:

WR1: there shall not be any instance of **product_definition_relationship** that is not an instance of **next_assembly_usage_occurence**.

5.2.7.29 unique_ids_in_model

The rule **unique_ids_in_model** specifies that in a **SIR_model**, the values of the attribute id of **SIR_node**s, respectively **SIR_node_relationship**s, respectively **SIR_submodel_usage**s shall be unique for each category of entities.

EXPRESS specification:

```
*)
RULE unique_ids_in_model FOR (SIR_model, SIR_node, SIR_node_relationship,
SIR_submodel_usage);
WHERE
      WR1:SIZEOF(QUERY(sm <* SIR_model | NOT (unique_ids(QUERY(sn<* SIR_node | sn
IN sm\representation.items)) AND unique_ids(QUERY(snr<* SIR_node_relationship |
snr IN sm\representation.items)) AND unique_ids(QUERY(su<* SIR_submodel_usage | su
IN sm\representation.items))) )) =0;
END_RULE;
(*
```

Argument definitions:

SIR_model: specifies the set of all instances of **SIR_model**.

SIR_node: specifies the set of all instances of **SIR_node**.

SIR_node_relationship: specifies the set of all instances of **SIR_node_relationship**.

SIR_submodel_usage: specifies the set of all instances of **SIR_submodel_usage**.

Formal propositions:

WR1: there shall not any instance of **SIR_model** such that the identifiers of the **SIR_node** (resp. **SIR_node_relationship**, resp. **SIR_submodel_usage**) that are elements of the **SIR_model** are not unique.

5.2.8 NRF_schema function definitions

5.2.8.1 get_names

The function **get_names** computes the set of **label**s assigned as a name to a **named_item** through **NRF_name_assignment.**

EXPRESS specification:

```
*)
FUNCTION get_names(prop: named_item): SET OF label;
LOCAL
      i: INTEGER;
      sav: BAG OF NRF_name_assignment;
      result: SET OF label;
END_LOCAL;

sav := USEDIN( prop, 'NRF_SCHEMA.NRF_NAME_ASSIGNMENT.ITEMS');

REPEAT i:=LOINDEX(sav) TO HIINDEX(sav);
      result[i]:=sav[i].assigned_name;
END_REPEAT;

RETURN(result);
END_FUNCTION;
(*
```

Argument definition:

prop: specifies the **named_item** for which the names are computed.

5.2.8.2 get_tools

The function **get_tools** computes the set of **action_resource**s that refer to a **NRF_run** and having the kind 'tool_or_facility'**.**

EXPRESS specification:

```
*)
FUNCTION get_tools(prop: NRF_run): SET OF action_resource;
LOCAL
      i: INTEGER;
      sav: BAG OF action_resource;
      result: SET OF action_resource;
END_LOCAL;

sav := USEDIN( prop, 'NRF_SCHEMA.ACTION_RESOURCE.USAGE');

REPEAT i:=LOINDEX(sav) TO HIINDEX(sav);
      IF (sav[i].kind.name='tool_or_facility') THEN
      result[i]:=sav[i];
      END_IF;
END_REPEAT;

RETURN(result);
END_FUNCTION;
(*
```

Argument definition:

prop: specifies the **NRF_run** for which the creating tools and facilities are computed.

5.2.8.3 get_timestamps

The function **get_timestamps** computes the set of **date_and_time**s that are assigned through **NRF_date_assignment** to a **NRF_run** with the role 'start_date_and_time'**.**

EXPRESS specification:

```
*)
FUNCTION get_timestamps(prop: NRF_run): SET OF date_and_time;
LOCAL
      i: INTEGER;
      sav: BAG OF NRF_date_assignment;
      result: SET OF date_and_time;
END_LOCAL;

sav := USEDIN( prop, 'NRF_SCHEMA.NRF_DATE_ASSIGNMENT.ITEMS');

REPEAT i:=LOINDEX(sav) TO HIINDEX(sav);
      IF (sav[i].role.name='start_date_and_time') THEN
      result[i]:=sav[i].assigned_date_and_time;
      END_IF;
END_REPEAT;

RETURN(result);
END_FUNCTION;
(*
```

Argument definition:

prop: specifies the **NRF_run** for which the starting dates are computed.

5.2.8.4 unique_ids

The function **unique_ids** checks whether the attributes id of a bag of **SIR_model_constituent**s are unique**.**

The result is TRUE if the ids are unique.

EXPRESS specification:

```
*)
FUNCTION unique_ids(sav: BAG OF SIR_model_constituent): LOGICAL;
LOCAL
      i: INTEGER;
      bag_id: BAG OF identifier;
END_LOCAL;

REPEAT i:=LOINDEX(sav) TO HIINDEX(sav);
      bag_id[i] := sav[i].id;
END_REPEAT;
```

```
RETURN(VALUE_UNIQUE(bag_id));
END_FUNCTION;
(*
```

Argument definition:

sav: specifies the bag of **SIR_model_constituent** for which the unicity of the ids is checked.

```
*)
END_SCHEMA; -- NRF_schema
```

Clause 6 - Conformance Requirements

TBD

Annex A - AIM EXPRESS expanded listing (normative)

A.1 Introduction

The following EXPRESS is the expanded form of the short form schema given in subclause 5.2. In the event of any discrepancy between the short form and this expanded listing, the expanded listing shall be used.

This file was generated with EXTOOL Version 3.4, Express toolkit developed by Association GOSET.

A.2 AIM EXPRESS expanded listing

```
SCHEMA NRF_schema;

TYPE characterized_definition = SELECT
     ( characterized_product_definition
           );
   END_TYPE;

TYPE characterized_product_definition = SELECT
     (product_definition,
      product_definition_relationship);
   END_TYPE;

TYPE supported_item = SELECT
     (action,
      action_method);
   END_TYPE;

TYPE person_organization_select = SELECT
     (person,
      organization,
      person_and_organization);
   END_TYPE;

TYPE date_time_select = SELECT
     (date,
      local_time,
      date_and_time);
   END_TYPE;

TYPE year_number = INTEGER;
   END_TYPE;

TYPE month_in_year_number = INTEGER;
   WHERE
     WR1: { 1 <= SELF <= 12 };
   END_TYPE;

TYPE day_in_month_number = INTEGER;
   END_TYPE;

TYPE ahead_or_behind = ENUMERATION OF
     (ahead,
      behind);
   END_TYPE;

TYPE hour_in_day = INTEGER;
   WHERE
     WR1: { 0 <= SELF < 24 };
```

```
   END_TYPE;

TYPE minute_in_hour = INTEGER;
   WHERE
     WR1: { 0 <= SELF <= 59 };
   END_TYPE;

TYPE second_in_minute = REAL;
   WHERE
     WR1: { 0 <= SELF < 60 };
   END_TYPE;

TYPE source_item = SELECT (identifier);
   END_TYPE;

TYPE identifier = STRING;
   END_TYPE;

TYPE label = STRING;
   END_TYPE;

TYPE text = STRING;
   END_TYPE;

TYPE measure_value = SELECT
     (length_measure,
      mass_measure,
      time_measure,
      electric_current_measure,
      thermodynamic_temperature_measure,
      amount_of_substance_measure,
      luminous_intensity_measure,
      plane_angle_measure,
      solid_angle_measure,
      area_measure,
      volume_measure,
      ratio_measure,
      parameter_value,
      numeric_measure,
      context_dependent_measure,
      descriptive_measure,
      positive_length_measure,
      positive_plane_angle_measure,
      positive_ratio_measure,
      count_measure);
   END_TYPE;

TYPE length_measure = REAL;
   END_TYPE;

TYPE mass_measure = REAL;
   END_TYPE;

TYPE time_measure = REAL;
   END_TYPE;

TYPE electric_current_measure = REAL;
   END_TYPE;

TYPE thermodynamic_temperature_measure = REAL;
   END_TYPE;

TYPE amount_of_substance_measure = REAL;
   END_TYPE;

TYPE luminous_intensity_measure = REAL;
   END_TYPE;
```

```
TYPE plane_angle_measure = REAL;
   END_TYPE;

TYPE solid_angle_measure = REAL;
   END_TYPE;

TYPE area_measure = REAL;
   END_TYPE;

TYPE volume_measure = REAL;
   END_TYPE;

TYPE ratio_measure = REAL;
   END_TYPE;

TYPE parameter_value = REAL;
   END_TYPE;

TYPE numeric_measure = NUMBER;
END_TYPE;

TYPE positive_length_measure = length_measure;
WHERE
      WR1: SELF > 0;
END_TYPE;

TYPE positive_plane_angle_measure = plane_angle_measure;
   WHERE
     WR1: SELF > 0;
   END_TYPE;

TYPE positive_ratio_measure = ratio_measure;
   WHERE
     WR1: SELF > 0;
   END_TYPE;

TYPE context_dependent_measure = REAL;
   END_TYPE;

TYPE descriptive_measure = STRING;
   END_TYPE;

TYPE count_measure = NUMBER;
   END_TYPE;

TYPE unit = SELECT
     (named_unit,
      derived_unit);
   END_TYPE;

TYPE si_unit_name = ENUMERATION OF
     (metre,
      gram,
      second,
      ampere,
      kelvin,
      mole,
      candela,
      radian,
      steradian,
      hertz,
      newton,
      pascal,
      joule,
      watt,
      coulomb,
      volt,
      farad,
```

```
      ohm,
      siemens,
      weber,
      tesla,
      henry,
      degree_Celsius,
      lumen,
      lux,
      becquerel,
      gray,
      sievert);
   END_TYPE;

TYPE si_prefix = ENUMERATION OF
     (exa,
      peta,
      tera,
      giga,
      mega,
      kilo,
      hecto,
      deca,
      deci,
      centi,
      milli,
      micro,
      nano,
      pico,
      femto,
      atto);
   END_TYPE;

TYPE dimension_count = INTEGER;
WHERE
  WR1: SELF > 0;
END_TYPE;

TYPE less_or_greater = SELECT(

           comparison_less_equal,

           comparison_greater_equal);

END_TYPE;

TYPE tabular_interpolation = ENUMERATION OF (

           polynomial,

           linear_logarithmic);

END_TYPE;

TYPE tabular_interpolation_degree = INTEGER;
WHERE
      WR1: SELF>=1;
END_TYPE;

TYPE list_real_literals = LIST[1:?] OF REAL;

END_TYPE;

TYPE list_ascending_real_literals = list_real_literals;

WHERE

      WR1: ascending_list(SELF);
```

```
END_TYPE;

TYPE SIR_model_constituent = SELECT(

            SIR_node,

            SIR_node_relationship,

            SIR_submodel_usage

            );

END_TYPE;

TYPE SIR_node_or_usage = SELECT(

            SIR_node,

            SIR_node_usage

            );

END_TYPE;

TYPE SIR_network_component = SELECT(

            SIR_model,

            SIR_node,

            SIR_node_usage,

            SIR_node_relationship,

            SIR_node_relationship_usage,

            SIR_submodel_usage

            );

END_TYPE;

TYPE SIR_property_symmetry = ENUMERATION OF (symmetrical, antisymmetrical);

END_TYPE;

TYPE SIR_list_of_values_or_functions = SELECT (

            SIR_list_of_descriptive_values, SIR_list_of_functions);

END_TYPE;

TYPE SIR_descriptive_or_functional = SELECT (
            SIR_property_descriptive, SIR_property_functional);
END_TYPE;

TYPE SIR_lifetime_type = ENUMERATION OF (invariant, sample, interval);

END_TYPE ;

TYPE SIR_abscissa_sequencing_type = ENUMERATION OF (

            strictly_decreasing,

            monotonic_decreasing,

            monotonic_increasing,
```

```
            strictly_increasing

            );

END_TYPE ;

TYPE SIR_case_or_phase = SELECT (

            SIR_ato_case, SIR_ato_phase);

END_TYPE;

TYPE SIR_number= NUMBER;
END_TYPE;

TYPE SIR_string= STRING;
END_TYPE;

TYPE SIR_number_or_string = SELECT (
            SIR_number,
            SIR_string);
END_TYPE;

TYPE SIR_index_interval = ARRAY[1:2] OF INTEGER;

WHERE

      WR1: SELF[1] >= 0;

      WR2: SELF[2] >= SELF[1];

END_TYPE;

ENTITY application_context;
     application          : text;
   INVERSE
     context_elements : SET [1:?] OF  application_context_element
                                   FOR frame_of_reference;
   END_ENTITY;

ENTITY application_protocol_definition;
     status                                    : label;
     application_interpreted_model_schema_name : label;
     application_protocol_year                 : year_number;
     application                               : application_context;
   END_ENTITY;

ENTITY application_context_element
     SUPERTYPE OF (ONEOF (product_context,
                          product_definition_context));
     name                 : label;
     frame_of_reference   : application_context;
   END_ENTITY;

ENTITY product_context
     SUBTYPE OF (application_context_element);
     discipline_type : label;
   END_ENTITY;

ENTITY product_definition_context
     SUBTYPE OF (application_context_element);
     life_cycle_stage : label;
   END_ENTITY;

ENTITY product;
     id                : identifier;
     name              : label;
     description       : text;
```

```
     frame_of_reference : SET [1:?] OF product_context;
   UNIQUE
     UR1: id;
   END_ENTITY;

ENTITY product_definition_formation;
     id          : identifier;
     description : text;
     of_product  : product;
   UNIQUE
     UR1: id, of_product;
   END_ENTITY;

ENTITY product_definition;
     id                : identifier;
     description       : text;
     formation         : product_definition_formation;
     frame_of_reference : product_definition_context;
   END_ENTITY;

ENTITY product_definition_relationship;
     id                          : identifier;
     name                        : label;
     description                 : text;
     relating_product_definition : product_definition;
     related_product_definition  : product_definition;
   END_ENTITY;

ENTITY property_definition;
     name        : label;
     description : text;
     definition  : characterized_definition;
   END_ENTITY;

ENTITY property_definition_representation;
     definition          : property_definition;
     used_representation : representation;
   END_ENTITY;

ENTITY name_assignment
     ABSTRACT SUPERTYPE;
     assigned_name : label;
   END_ENTITY;

ENTITY action_assignment
     ABSTRACT SUPERTYPE;
     assigned_action : action;
   END_ENTITY;

ENTITY approval_assignment
     ABSTRACT SUPERTYPE;
     assigned_approval : approval;
   END_ENTITY;

ENTITY security_classification_assignment
     ABSTRACT SUPERTYPE;
     assigned_security_classification : security_classification;
   END_ENTITY;

ENTITY person_and_organization_assignment
     ABSTRACT SUPERTYPE;
     assigned_person_and_organization : person_and_organization;
     role                             : person_and_organization_role;
   END_ENTITY;

ENTITY date_and_time_assignment
     ABSTRACT SUPERTYPE;
     assigned_date_and_time : date_and_time;
```

```
    role                  : date_time_role;
  END_ENTITY;

ENTITY group_assignment
    ABSTRACT SUPERTYPE;
    assigned_group : group;
  END_ENTITY;

ENTITY document_type;
    product_data_type : label;
  END_ENTITY;

ENTITY document;
    id          : identifier;
    name        : label;
    description : text;
    kind        : document_type;
  UNIQUE
    UR1: id;
  END_ENTITY;

ENTITY document_usage_constraint;
    source               : document;
    subject_element      : label;
    subject_element_value : text;
  END_ENTITY;

ENTITY action;
    name          : label;
    description   : text;
    chosen_method : action_method;
  END_ENTITY;

ENTITY executed_action
    SUBTYPE OF (action);
  END_ENTITY;

ENTITY action_relationship;
    name            : label;
    description     : text;
    relating_action : action;
    related_action  : action;
  END_ENTITY;

ENTITY action_method;
    name        : label;
    description : text;
    consequence : text;
    purpose     : text;
  END_ENTITY;

ENTITY action_resource;
    name         : label;
    description  : text;
    usage        : supported_item;
    kind         : action_resource_type;
  END_ENTITY;

ENTITY action_resource_type;
    name : label;
  END_ENTITY;

ENTITY approval_status;
    name : label;
  END_ENTITY;

ENTITY approval;
    status : approval_status;
```

```
    level  : label;
  END_ENTITY;

ENTITY approval_date_time;
    date_time      : date_time_select;
    dated_approval : approval;
  END_ENTITY;

ENTITY approval_person_organization;
    person_organization : person_organization_select;
    authorized_approval : approval;
    role                : approval_role;
  END_ENTITY;

ENTITY approval_role;
    role : label;
  END_ENTITY;

ENTITY security_classification_level;
    name : label;
  END_ENTITY;

ENTITY security_classification;
    name           : label;
    purpose        : text;
    security_level : security_classification_level;
  END_ENTITY;

ENTITY address;
    internal_location       : OPTIONAL label;
    street_number           : OPTIONAL label;
    street                  : OPTIONAL label;
    postal_box              : OPTIONAL label;
    town                    : OPTIONAL label;
    region                  : OPTIONAL label;
    postal_code             : OPTIONAL label;
    country                 : OPTIONAL label;
    facsimile_number        : OPTIONAL label;
    telephone_number        : OPTIONAL label;
    electronic_mail_address : OPTIONAL label;
    telex_number            : OPTIONAL label;
  WHERE
    WR1: EXISTS(internal_location)       OR
         EXISTS(street_number)           OR
         EXISTS(street)                  OR
         EXISTS(postal_box)              OR
         EXISTS(town)                    OR
         EXISTS(region)                  OR
         EXISTS(postal_code)             OR
         EXISTS(country)                 OR
         EXISTS(facsimile_number)        OR
         EXISTS(telephone_number)        OR
         EXISTS(electronic_mail_address) OR
         EXISTS(telex_number);
  END_ENTITY;

ENTITY personal_address
    SUBTYPE OF (address);
    people      : SET [1:?] OF person;
    description : text;
  END_ENTITY;

ENTITY organizational_address
    SUBTYPE OF (address);
    organizations : SET [1:?] OF organization;
    description   : text;
  END_ENTITY;
```

```
ENTITY person;
     id            : identifier;
     last_name     : OPTIONAL label;
     first_name    : OPTIONAL label;
     middle_names  : OPTIONAL LIST [1:?] OF label;
     prefix_titles : OPTIONAL LIST [1:?] OF label;
     suffix_titles : OPTIONAL LIST [1:?] OF label;
   UNIQUE
     UR1: id;
   WHERE
     WR1: EXISTS(last_name) OR EXISTS(first_name);
   END_ENTITY;

ENTITY organization;
     id          : OPTIONAL identifier;
     name        : label;
     description : text;
   END_ENTITY;

ENTITY organizational_project;
     name                       : label;
     description                : text;
     responsible_organizations : SET[1:?] OF organization;
   END_ENTITY;

ENTITY person_and_organization;
     the_person       : person;
     the_organization : organization;
   END_ENTITY;

ENTITY person_and_organization_role;
     name : label;
   END_ENTITY;

ENTITY date
     SUPERTYPE OF (ONEOF (calendar_date));
     year_component : year_number;
   END_ENTITY;

ENTITY calendar_date
     SUBTYPE OF (date);
     day_component   : day_in_month_number;
     month_component : month_in_year_number;
   WHERE
     WR1: valid_calendar_date (SELF);
   END_ENTITY;

ENTITY coordinated_universal_time_offset;
     hour_offset   : hour_in_day;
     minute_offset : OPTIONAL minute_in_hour;
     sense         : ahead_or_behind;
   END_ENTITY;

ENTITY local_time;
     hour_component   : hour_in_day;
     minute_component : OPTIONAL minute_in_hour;
     second_component : OPTIONAL second_in_minute;
     zone             : coordinated_universal_time_offset;
   WHERE
     WR1: valid_time (SELF);
   END_ENTITY;

ENTITY date_and_time;
     date_component : date;
     time_component : local_time;
   END_ENTITY;

ENTITY date_time_role;
```

```
     name : label;
   END_ENTITY;

ENTITY group;
     name        : label;
     description : text;
   END_ENTITY;

ENTITY named_unit
     SUPERTYPE OF (ONEOF (si_unit, conversion_based_unit,
context_dependent_unit)
                   ANDOR
                   ONEOF (length_unit,
                          mass_unit,
                          time_unit,
                          electric_current_unit,
                          thermodynamic_temperature_unit,
                          amount_of_substance_unit,
                          luminous_intensity_unit,
                          plane_angle_unit,
                          solid_angle_unit,
                          area_unit,
                          volume_unit,
                          ratio_unit ));
     dimensions : dimensional_exponents;
   END_ENTITY;

ENTITY si_unit
     SUBTYPE OF (named_unit);
     prefix     : OPTIONAL si_prefix;
     name       : si_unit_name;
   DERIVE
     SELF\named_unit.dimensions : dimensional_exponents
                                := dimensions_for_si_unit (SELF.name);
   END_ENTITY;

ENTITY conversion_based_unit
     SUBTYPE OF (named_unit);
     name              : label;
     conversion_factor : measure_with_unit;
   END_ENTITY;

ENTITY context_dependent_unit
     SUBTYPE OF (named_unit);
     name : label;
   END_ENTITY;

ENTITY length_unit
     SUBTYPE OF (named_unit);
   WHERE
     WR1: (SELF\named_unit.dimensions.length_exponent = 1.0)
    AND
          (SELF\named_unit.dimensions.mass_exponent = 0.0)
    AND
          (SELF\named_unit.dimensions.time_exponent = 0.0)
    AND
          (SELF\named_unit.dimensions.electric_current_exponent = 0.0)
    AND
          (SELF\named_unit.dimensions.thermodynamic_temperature_exponent =
0.0) AND
          (SELF\named_unit.dimensions.amount_of_substance_exponent = 0.0)
AND
          (SELF\named_unit.dimensions.luminous_intensity_exponent = 0.0);
   END_ENTITY;

ENTITY mass_unit
     SUBTYPE OF (named_unit);
   WHERE
```

```
     WR1: (SELF\named_unit.dimensions.length_exponent                    =
0.0) AND
          (SELF\named_unit.dimensions.mass_exponent                      =
1.0) AND
          (SELF\named_unit.dimensions.time_exponent                      =
0.0) AND
          (SELF\named_unit.dimensions.electric_current_exponent          =
0.0) AND
          (SELF\named_unit.dimensions.thermodynamic_temperature_exponent =
0.0) AND
          (SELF\named_unit.dimensions.amount_of_substance_exponent       =
0.0) AND
          (SELF\named_unit.dimensions.luminous_intensity_exponent        =
0.0);
   END_ENTITY ;

ENTITY time_unit
     SUBTYPE OF (named_unit);
   WHERE
     WR1: (SELF\named_unit.dimensions.length_exponent                    =
0.0) AND
          (SELF\named_unit.dimensions.mass_exponent                      =
0.0) AND
          (SELF\named_unit.dimensions.time_exponent                      =
1.0) AND
          (SELF\named_unit.dimensions.electric_current_exponent          =
0.0) AND
          (SELF\named_unit.dimensions.thermodynamic_temperature_exponent =
0.0) AND
          (SELF\named_unit.dimensions.amount_of_substance_exponent       =
0.0) AND
          (SELF\named_unit.dimensions.luminous_intensity_exponent        =
0.0);
   END_ENTITY;

ENTITY electric_current_unit
     SUBTYPE OF (named_unit);
   WHERE
     WR1: (SELF\named_unit.dimensions.length_exponent                    =
0.0) AND
          (SELF\named_unit.dimensions.mass_exponent                      =
0.0) AND
          (SELF\named_unit.dimensions.time_exponent                      =
0.0) AND
          (SELF\named_unit.dimensions.electric_current_exponent          =
1.0) AND
          (SELF\named_unit.dimensions.thermodynamic_temperature_exponent =
0.0) AND
          (SELF\named_unit.dimensions.amount_of_substance_exponent       =
0.0) AND
          (SELF\named_unit.dimensions.luminous_intensity_exponent        =
0.0);
   END_ENTITY;

ENTITY thermodynamic_temperature_unit
     SUBTYPE OF (named_unit);
   WHERE
     WR1: (SELF\named_unit.dimensions.length_exponent                    =
0.0) AND
          (SELF\named_unit.dimensions.mass_exponent                      =
0.0) AND
          (SELF\named_unit.dimensions.time_exponent                      =
0.0) AND
          (SELF\named_unit.dimensions.electric_current_exponent          =
0.0) AND
          (SELF\named_unit.dimensions.thermodynamic_temperature_exponent =
1.0) AND
          (SELF\named_unit.dimensions.amount_of_substance_exponent       =
```

```
0.0) AND
          (SELF\named_unit.dimensions.luminous_intensity_exponent        =
0.0);
   END_ENTITY;

ENTITY amount_of_substance_unit
     SUBTYPE OF (named_unit);
   WHERE
     WR1: (SELF\named_unit.dimensions.length_exponent                     =
0.0) AND
               (SELF\named_unit.dimensions.mass_exponent
= 0.0) AND
               (SELF\named_unit.dimensions.time_exponent
= 0.0) AND
               (SELF\named_unit.dimensions.electric_current_exponent
= 0.0) AND
               (SELF\named_unit.dimensions.thermodynamic_temperature_exponent
= 0.0) AND
               (SELF\named_unit.dimensions.amount_of_substance_exponent
= 1.0) AND
               (SELF\named_unit.dimensions.luminous_intensity_exponent
= 0.0);
   END_ENTITY;

ENTITY luminous_intensity_unit
     SUBTYPE OF (named_unit);
   WHERE
     WR1: (SELF\named_unit.dimensions.length_exponent                     =
0.0) AND
          (SELF\named_unit.dimensions.mass_exponent                       =
0.0) AND
          (SELF\named_unit.dimensions.time_exponent                       =
0.0) AND
          (SELF\named_unit.dimensions.electric_current_exponent           =
0.0) AND
          (SELF\named_unit.dimensions.thermodynamic_temperature_exponent =
0.0) AND
          (SELF\named_unit.dimensions.amount_of_substance_exponent        =
0.0) AND
          (SELF\named_unit.dimensions.luminous_intensity_exponent         =
1.0);
   END_ENTITY;

ENTITY plane_angle_unit
     SUBTYPE OF (named_unit);
   WHERE
     WR1: (SELF\named_unit.dimensions.length_exponent                     =
0.0) AND
          (SELF\named_unit.dimensions.mass_exponent                       =
0.0) AND
          (SELF\named_unit.dimensions.time_exponent                       =
0.0) AND
          (SELF\named_unit.dimensions.electric_current_exponent           =
0.0) AND
          (SELF\named_unit.dimensions.thermodynamic_temperature_exponent =
0.0) AND
          (SELF\named_unit.dimensions.amount_of_substance_exponent        =
0.0) AND
          (SELF\named_unit.dimensions.luminous_intensity_exponent         =
0.0);
   END_ENTITY;

ENTITY solid_angle_unit
     SUBTYPE OF (named_unit);
   WHERE
     WR1: (SELF\named_unit.dimensions.length_exponent                     =
0.0) AND
          (SELF\named_unit.dimensions.mass_exponent                       =
```

```
0.0) AND
          (SELF\named_unit.dimensions.time_exponent                      =
0.0) AND
          (SELF\named_unit.dimensions.electric_current_exponent          =
0.0) AND
          (SELF\named_unit.dimensions.thermodynamic_temperature_exponent =
0.0) AND
          (SELF\named_unit.dimensions.amount_of_substance_exponent       =
0.0) AND
          (SELF\named_unit.dimensions.luminous_intensity_exponent        =
0.0);
   END_ENTITY;

ENTITY area_unit
     SUBTYPE OF (named_unit);
   WHERE
     WR1: (SELF\named_unit.dimensions.length_exponent = 2.0)
    AND
          (SELF\named_unit.dimensions.mass_exponent = 0.0)
    AND
          (SELF\named_unit.dimensions.time_exponent = 0.0)
    AND
          (SELF\named_unit.dimensions.electric_current_exponent = 0.0)
    AND
          (SELF\named_unit.dimensions.thermodynamic_temperature_exponent =
0.0) AND
          (SELF\named_unit.dimensions.amount_of_substance_exponent = 0.0)
    AND
          (SELF\named_unit.dimensions.luminous_intensity_exponent = 0.0);
   END_ENTITY;

ENTITY volume_unit
     SUBTYPE OF (named_unit);
   WHERE
     WR1: (SELF\named_unit.dimensions.length_exponent = 3.0)
    AND
          (SELF\named_unit.dimensions.mass_exponent = 0.0)
    AND
          (SELF\named_unit.dimensions.time_exponent = 0.0)
    AND
          (SELF\named_unit.dimensions.electric_current_exponent = 0.0)
    AND
          (SELF\named_unit.dimensions.thermodynamic_temperature_exponent =
0.0) AND
          (SELF\named_unit.dimensions.amount_of_substance_exponent = 0.0)
    AND
          (SELF\named_unit.dimensions.luminous_intensity_exponent = 0.0);
   END_ENTITY;

ENTITY ratio_unit
     SUBTYPE OF (named_unit);
   WHERE
     WR1: (SELF\named_unit.dimensions.length_exponent = 0.0)
    AND
          (SELF\named_unit.dimensions.mass_exponent = 0.0)
    AND
          (SELF\named_unit.dimensions.time_exponent = 0.0)
    AND
          (SELF\named_unit.dimensions.electric_current_exponent = 0.0)
    AND
          (SELF\named_unit.dimensions.thermodynamic_temperature_exponent =
0.0) AND
          (SELF\named_unit.dimensions.amount_of_substance_exponent = 0.0)
    AND
          (SELF\named_unit.dimensions.luminous_intensity_exponent = 0.0);
   END_ENTITY;

ENTITY dimensional_exponents;
```

```
     length_exponent                    : REAL;
     mass_exponent                      : REAL;
     time_exponent                      : REAL;
     electric_current_exponent          : REAL;
     thermodynamic_temperature_exponent : REAL;
     amount_of_substance_exponent       : REAL;
     luminous_intensity_exponent        : REAL;
   END_ENTITY;

ENTITY derived_unit_element;
     unit     : named_unit;
     exponent : REAL;
   END_ENTITY;

ENTITY derived_unit;
     elements : SET [1:?] OF derived_unit_element;
   WHERE
     WR1 : ( SIZEOF ( elements ) > 1 ) OR
             (( SIZEOF ( elements ) = 1 ) AND ( elements[1].exponent <> 1.0
));
   END_ENTITY;

ENTITY measure_with_unit
     SUPERTYPE OF (ONEOF ( length_measure_with_unit,
                           mass_measure_with_unit,
                           time_measure_with_unit,
                           electric_current_measure_with_unit,
                           thermodynamic_temperature_measure_with_unit,
                           amount_of_substance_measure_with_unit,
                           luminous_intensity_measure_with_unit,
                           plane_angle_measure_with_unit,
                           solid_angle_measure_with_unit,
                           area_measure_with_unit,
                           volume_measure_with_unit,
                           ratio_measure_with_unit ));
     value_component : measure_value;
     unit_component  : unit;
   WHERE
     WR1: valid_units (SELF);
   END_ENTITY;

ENTITY length_measure_with_unit
     SUBTYPE OF (measure_with_unit);
   WHERE
     WR1: 'NRF_SCHEMA.LENGTH_UNIT' IN TYPEOF
(SELF\measure_with_unit.unit_component);
   END_ENTITY;

ENTITY mass_measure_with_unit
     SUBTYPE OF (measure_with_unit);
   WHERE
     WR1: 'NRF_SCHEMA.MASS_UNIT' IN TYPEOF
(SELF\measure_with_unit.unit_component);
   END_ENTITY;

ENTITY time_measure_with_unit
     SUBTYPE OF (measure_with_unit);
   WHERE
     WR1: 'NRF_SCHEMA.TIME_UNIT' IN TYPEOF
(SELF\measure_with_unit.unit_component);
   END_ENTITY;

ENTITY electric_current_measure_with_unit
     SUBTYPE OF (measure_with_unit);
   WHERE
     WR1: 'NRF_SCHEMA.ELECTRIC_CURRENT_UNIT' IN TYPEOF
(SELF\measure_with_unit.unit_component);
   END_ENTITY;
```

```
ENTITY thermodynamic_temperature_measure_with_unit
     SUBTYPE OF (measure_with_unit);
   WHERE
     WR1: 'NRF_SCHEMA.THERMODYNAMIC_TEMPERATURE_UNIT' IN
          TYPEOF (SELF\measure_with_unit.unit_component);
   END_ENTITY;

ENTITY amount_of_substance_measure_with_unit
     SUBTYPE OF (measure_with_unit);
   WHERE
     WR1: 'NRF_SCHEMA.AMOUNT_OF_SUBSTANCE_UNIT' IN
          TYPEOF (SELF\measure_with_unit.unit_component);
   END_ENTITY;

ENTITY luminous_intensity_measure_with_unit
     SUBTYPE OF (measure_with_unit);
   WHERE
     WR1: 'NRF_SCHEMA.LUMINOUS_INTENSITY_UNIT' IN
          TYPEOF (SELF\measure_with_unit.unit_component);
   END_ENTITY;

ENTITY plane_angle_measure_with_unit
     SUBTYPE OF (measure_with_unit);
   WHERE
     WR1: 'NRF_SCHEMA.PLANE_ANGLE_UNIT' IN
          TYPEOF (SELF\measure_with_unit.unit_component);
   END_ENTITY;

ENTITY solid_angle_measure_with_unit
     SUBTYPE OF (measure_with_unit);
   WHERE
     WR1: 'NRF_SCHEMA.SOLID_ANGLE_UNIT' IN TYPEOF
(SELF\measure_with_unit.unit_component);
   END_ENTITY;

ENTITY area_measure_with_unit
     SUBTYPE OF (measure_with_unit);
   WHERE
     WR1: 'NRF_SCHEMA.AREA_UNIT' IN TYPEOF
(SELF\measure_with_unit.unit_component);
   END_ENTITY;

ENTITY volume_measure_with_unit
     SUBTYPE OF (measure_with_unit);
   WHERE
     WR1: 'NRF_SCHEMA.VOLUME_UNIT' IN TYPEOF
(SELF\measure_with_unit.unit_component);
   END_ENTITY;

ENTITY ratio_measure_with_unit
     SUBTYPE OF (measure_with_unit);
   WHERE
     WR1: 'NRF_SCHEMA.RATIO_UNIT' IN TYPEOF
(SELF\measure_with_unit.unit_component);
   END_ENTITY;

ENTITY representation_context;
  context_identifier : identifier;
  context_type       : text;
INVERSE
  representations_in_context : SET [1:?] OF representation
    FOR context_of_items;
END_ENTITY;

ENTITY representation_item;
  name : label;
WHERE
```

```
  WR1: SIZEOF(using_representations(SELF)) > 0;
END_ENTITY;

ENTITY representation;
  name             : label;
  items            : SET[1:?] OF representation_item;
  context_of_items : representation_context;
END_ENTITY;

ENTITY representation_map;
  mapping_origin        : representation_item;
  mapped_representation : representation;
INVERSE
  map_usage : SET[1:?] OF mapped_item FOR mapping_source;
WHERE
  WR1: item_in_context(SELF.mapping_origin,
       SELF.mapped_representation.context_of_items);
END_ENTITY;

ENTITY mapped_item
  SUBTYPE OF (representation_item);
  mapping_source : representation_map;
  mapping_target : representation_item;
WHERE
  WR1: acyclic_mapped_representation(using_representations(SELF), [SELF]);
END_ENTITY;

ENTITY product_definition_usage
  SUPERTYPE OF (ONEOF (assembly_component_usage))
  SUBTYPE OF (product_definition_relationship);
UNIQUE
  UR1: SELF\product_definition_relationship.id,
       SELF\product_definition_relationship.relating_product_definition,
       SELF\product_definition_relationship.related_product_definition;
WHERE
  WR1:  acyclic_product_definition_relationship
         (SELF,
         [SELF\product_definition_relationship.related_product_definition],
         'NRF_SCHEMA.PRODUCT_DEFINITION_USAGE.' +
         'RELATED_PRODUCT_DEFINITION');
END_ENTITY;

ENTITY assembly_component_usage
  SUPERTYPE OF (ONEOF (next_assembly_usage_occurrence))
  SUBTYPE OF (product_definition_usage);
  reference_designator : OPTIONAL identifier;
END_ENTITY;

ENTITY next_assembly_usage_occurrence
  SUBTYPE OF (assembly_component_usage);
END_ENTITY;

ENTITY type_qualifier;
  name : label;
END_ENTITY;

ENTITY generic_expression
ABSTRACT SUPERTYPE OF(ONEOF(simple_gen_expression,
                 unary_gen_expression,
                 binary_gen_expression,
                 m_ary_gen_expression));
WHERE
     WR1: is_acyclic(SELF);
END_ENTITY;

ENTITY simple_gen_expression
ABSTRACT SUPERTYPE OF (ONEOF(generic_literal, generic_variable))
SUBTYPE OF (generic_expression);
```

```
END_ENTITY;

ENTITY generic_literal
ABSTRACT SUPERTYPE
SUBTYPE OF (simple_gen_expression);
END_ENTITY;

ENTITY generic_variable
ABSTRACT SUPERTYPE
SUBTYPE OF (simple_gen_expression);
INVERSE
      interpretation:SET [0:1] OF environment FOR syntactic_representation;
END_ENTITY;

ENTITY variable_semantics
ABSTRACT SUPERTYPE;
END_ENTITY;

ENTITY environment;
      syntactic_representation:generic_variable;
      semantics  :     variable_semantics;
END_ENTITY;

ENTITY unary_gen_expression
ABSTRACT SUPERTYPE
SUBTYPE OF(generic_expression);
      operand : generic_expression;
END_ENTITY;

ENTITY binary_gen_expression
ABSTRACT SUPERTYPE
SUBTYPE OF(generic_expression);
operands : LIST [2:2] OF generic_expression;
END_ENTITY;

ENTITY m_ary_gen_expression
ABSTRACT SUPERTYPE
SUBTYPE OF(generic_expression);
operands : LIST [2:?] OF generic_expression;
END_ENTITY;

ENTITY expression
      ABSTRACT SUPERTYPE OF (ONEOF (numeric_expression,
                        boolean_expression))
      SUBTYPE OF (generic_expression);
END_ENTITY;

ENTITY variable
      ABSTRACT SUPERTYPE OF (ONEOF (numeric_variable))
      SUBTYPE OF(generic_variable);
END_ENTITY;

ENTITY numeric_expression
      ABSTRACT SUPERTYPE OF (ONEOF (simple_n_expression,
                        unary_n_expression,
                        binary_n_expression,
                        m_ary_n_expression))
      SUBTYPE OF (expression);
DERIVE
      is_int: BOOLEAN := is_int_expr (SELF);
      sql_mappable: BOOLEAN := is_SQL_mappable (SELF);
END_ENTITY;

ENTITY simple_n_expression
      ABSTRACT SUPERTYPE OF (ONEOF (literal_number, numeric_variable))
      SUBTYPE OF (numeric_expression, simple_gen_expression);
END_ENTITY;
```

```
ENTITY literal_number
      ABSTRACT SUPERTYPE OF (ONEOF (int_literal, real_literal))
      SUBTYPE OF (simple_n_expression, generic_literal);
      the_value: NUMBER;
END_ENTITY;

ENTITY int_literal
      SUBTYPE OF (literal_number);
      SELF\literal_number.the_value: INTEGER;
END_ENTITY;

ENTITY real_literal
      SUBTYPE OF (literal_number);
      SELF\literal_number.the_value: REAL;
END_ENTITY;

ENTITY numeric_variable
      SUPERTYPE OF (ONEOF (int_numeric_variable,
                           real_numeric_variable))
      SUBTYPE OF (simple_n_expression, variable);
WHERE
WR1: ‘NRF_SCHEMA.INT_NUMERIC_VARIABLE'
            IN TYPEOF(SELF) ) OR
            ('NRF_SCHEMA.REAL_NUMERIC_VARIABLE'
            IN TYPEOF(SELF) );
END_ENTITY;

ENTITY int_numeric_variable
      SUBTYPE OF (numeric_variable);
END_ENTITY;

ENTITY real_numeric_variable
      SUBTYPE OF (numeric_variable);
END_ENTITY;

ENTITY unary_n_expression
      SUBTYPE OF (numeric_expression, unary_gen_expression);
      SELF\unary_gen_expression.operand: numeric_expression;
END_ENTITY;

ENTITY binary_n_expression
      ABSTRACT SUPERTYPE OF (ONEOF (power_expression))
      SUBTYPE OF (numeric_expression, binary_gen_expression);
      SELF\binary_gen_expression.operands : LIST [2:2] OF
                        numeric_expression;
END_ENTITY;

ENTITY m_ary_n_expression
      ABSTRACT SUPERTYPE OF (ONEOF (plus_expression,
                        mult_expression))
      SUBTYPE OF (numeric_expression, m_ary_gen_expression);
      SELF\m_ary_gen_expression.operands: LIST [2:?] OF
                        numeric_expression;
END_ENTITY;

ENTITY plus_expression
      SUBTYPE OF (m_ary_n_expression);
END_ENTITY;

ENTITY mult_expression
      SUBTYPE OF (m_ary_n_expression);
END_ENTITY;

ENTITY power_expression
      SUBTYPE OF (binary_n_expression);
END_ENTITY;

ENTITY boolean_expression
```

```
      ABSTRACT SUPERTYPE OF (ONEOF (comparison_expression))
      SUBTYPE OF (expression);
END_ENTITY;

ENTITY comparison_expression
      ABSTRACT SUPERTYPE OF (ONEOF (comparison_greater_equal,
                        comparison_less_equal))
      SUBTYPE OF (boolean_expression, binary_gen_expression);
      SELF\binary_gen_expression.operands: LIST [2:2] OF
generic_expression;
END_ENTITY;

ENTITY comparison_greater_equal
      SUBTYPE OF (comparison_expression);
END_ENTITY;

ENTITY comparison_less_equal
      SUBTYPE OF (comparison_expression);
END_ENTITY;

ENTITY mathematical_string;
text_representation: text;
SGML_representation: OPTIONAL text;
END_ENTITY;

ENTITY SIR_product_version

SUBTYPE OF (product_definition_formation);

WHERE

      WR1: SIZEOF(USEDIN(SELF,'NRF_SCHEMA' +
                  'PRODUCT_DEFINITION.FORMATION')) > 0;

END_ENTITY;

ENTITY SIR_variable

SUBTYPE OF (numeric_variable);

WHERE

      WR1:(SIZEOF(SELF\generic_variable.interpretation)=0) OR
(('NRF_SCHEMA.SIR_PROPERTY_USAGE' IN
TYPEOF(SELF\generic_variable.interpretation[1].semantics)) AND
('NRF_SCHEMA.SIR_PROPERTY_QUANTITATIVE' IN
TYPEOF(SELF\generic_variable.interpretation[1].semantics.property)));

END_ENTITY;

ENTITY SIR_parameterized_function

ABSTRACT SUPERTYPE OF (ONEOF( SIR_polynomial_function, SIR_tabular_function));

      name: label;

      description: text;

      formula: OPTIONAL mathematical_string;

      parameters: LIST[1:?] OF UNIQUE SIR_variable;

      result: SIR_variable;

END_ENTITY;

ENTITY SIR_polynomial_function
```

```
SUBTYPE OF (SIR_parameterized_function);

      lower_bound_expressions: SET[0:?] OF comparison_less_equal;

      upper_bound_expressions: SET[0:?] OF comparison_greater_equal;

      polynom_expression: numeric_expression;

WHERE

     WR1:valid_bounded_variables(lower_bound_expressions,
SELF\SIR_parameterized_function.parameters);

     WR2:valid_bounded_variables(upper_bound_expressions,
SELF\SIR_parameterized_function.parameters);

END_ENTITY;

ENTITY SIR_tabular_function

SUBTYPE OF (SIR_parameterized_function);

      parameter_values: LIST[1:?] OF list_ascending_real_literals;

      result_values: list_real_literals;

      interpolation_type: tabular_interpolation;

      interpolation_degree: OPTIONAL tabular_interpolation_degree;

WHERE

     WR1: ((interpolation_type=polynomial) AND EXISTS(interpolation_degree)) OR
(interpolation_type=linear_logarithmic);

     WR2: SIZEOF(SELF\SIR_parameterized_function.parameters) =
SIZEOF(parameter_values);

     WR3: consistent_size(parameter_values, result_values);

END_ENTITY;

ENTITY SIR_cyclic_tabular_function

SUBTYPE OF (SIR_tabular_function);

      period: real_literal;

WHERE

      WR1: SIZEOF(SELF\SIR_parameterized_function.parameters)=1;

END_ENTITY;

ENTITY SIR_model

SUBTYPE OF (representation);
      id: identifier;
      version_id: identifier;
      description: text;
      SELF\representation.items: SET[1:?] OF SIR_model_constituent;
UNIQUE
      UR1: id, version_id;
END_ENTITY;

ENTITY SIR_node

SUBTYPE OF (representation_item);
```

```
      id: identifier;
      classes: LIST[1:?] OF UNIQUE type_qualifier;

DERIVE

      item_of: BAG OF representation := USEDIN(SELF,
'NRF_SCHEMA.REPRESENTATION.ITEMS');

WHERE

      WR1:SIZEOF(QUERY(rep <* item_of|'NRF_SCHEMA.SIR_MODEL' IN TYPEOF(rep)))=1;

END_ENTITY;

ENTITY SIR_node_relationship

SUBTYPE OF (representation_item);

      id: identifier;
      classes: LIST[1:?] OF UNIQUE type_qualifier;

      nodes: LIST[2:?] OF SIR_node_or_usage;

DERIVE

      item_of: BAG OF representation := USEDIN(SELF,
'NRF_SCHEMA.REPRESENTATION.ITEMS');

WHERE

      WR1:SIZEOF(QUERY(rep <* item_of|'NRF_SCHEMA.SIR_MODEL' IN TYPEOF(rep)))=1;

END_ENTITY;

ENTITY SIR_submodel_usage

SUBTYPE OF (mapped_item);

      id: identifier;
DERIVE

      item_of: BAG OF representation := USEDIN(SELF,
'NRF_SCHEMA.REPRESENTATION.ITEMS');

WHERE

      WR1:'NRF_SCHEMA.SIR_MODEL' IN
TYPEOF(SELF\mapped_item.mapping_source.mapped_representation);

      WR2:('NRF_SCHEMA.SIR_NODE' IN
TYPEOF(SELF\mapped_item.mapping_source.mapping_origin)) AND (SELF IN
SELF\mapped_item.mapping_source.mapped_representation.items);

      WR3:'NRF_SCHEMA.SIR_NODE' IN TYPEOF(SELF\mapped_item.mapping_target);

      WR4:SIZEOF(QUERY(rep <* item_of|'NRF_SCHEMA.SIR_MODEL' IN TYPEOF(rep)))=1;

END_ENTITY;

ENTITY SIR_node_usage

SUBTYPE OF (representation_item);

      used_submodel: SIR_submodel_usage;

      component: SIR_node;
```

```
WHERE
      WR1: component IN used_submodel.mapping_source.mapped_representation.items;
END_ENTITY;
ENTITY SIR_node_relationship_usage
SUBTYPE OF (representation_item);
      used_submodel: SIR_submodel_usage;
      component: SIR_node_relationship;
WHERE
      WR1: component IN used_submodel.mapping_source.mapped_representation.items;
END_ENTITY;
ENTITY SIR_component_sequence;
      id: identifier;
      name: OPTIONAL label;
      components: LIST[1:?] OF UNIQUE SIR_network_component;
END_ENTITY;
ENTITY SIR_list_of_descriptive_values;
      name: label;
      entries: LIST[1:?] OF UNIQUE STRING;
INVERSE
      applicable_for: SET[1:?] OF SIR_applicability_of_values FOR valid_data;
END_ENTITY;
ENTITY SIR_list_of_functions;
      name: label;
      entries: LIST[1:?] OF UNIQUE SIR_parameterized_function;
INVERSE
      applicable_for: SET[1:?] OF SIR_applicability_of_values FOR valid_data;
END_ENTITY;
ENTITY SIR_property_name;
      name: label;
UNIQUE name;
END_ENTITY;
ENTITY SIR_property_class
ABSTRACT SUPERTYPE OF (ONEOF(SIR_property_scalar, SIR_property_tensor));
      name: SIR_property_name;
```

```
END_ENTITY;

ENTITY SIR_property_scalar

ABSTRACT SUPERTYPE OF (ONEOF(SIR_property_descriptive, SIR_property_quantitative,
SIR_property_functional))
SUBTYPE OF (SIR_property_class) ;

      symmetry: OPTIONAL SIR_property_symmetry;

END_ENTITY;

ENTITY SIR_property_descriptive

SUBTYPE OF (SIR_property_scalar) ;

DERIVE

      value_containers: SET OF SIR_list_of_descriptive_values :=
get_containers(SELF);
WHERE
      WR1: SIZEOF(value_containers)>=1;
END_ENTITY;

ENTITY SIR_property_functional

SUBTYPE OF (SIR_property_scalar) ;

DERIVE

      function_containers: SET OF SIR_list_of_functions := get_containers(SELF);
WHERE
      WR1: SIZEOF(function_containers)>=1;

END_ENTITY;

ENTITY SIR_applicability_of_values;

      valid_data: SIR_list_of_values_or_functions;

      valid_usage: SIR_property_usage;

WHERE

      WR1: (('NRF_SCHEMA.SIR_LIST_OF_DESCRIPTIVE_VALUES' IN TYPEOF(valid_data))
AND
('NRF_SCHEMA.SIR_PROPERTY_DESCRIPTIVE' IN TYPEOF(valid_usage.property))) OR
(('NRF_SCHEMA.SIR_LIST_OF_FUNCTIONS' IN TYPEOF(valid_data)) AND
('NRF_SCHEMA.SIR_PROPERTY_FUNCTIONAL' IN TYPEOF(valid_usage.property)));

END_ENTITY;

ENTITY SIR_property_quantitative

SUBTYPE OF (SIR_property_scalar) ;

END_ENTITY;

ENTITY SIR_property_tensor

SUBTYPE OF (SIR_property_class);

      tensor_order: INTEGER;

      dimensionality: INTEGER;

INVERSE
```

```
      members: SET[1:?] OF SIR_scalar_in_tensor FOR tensor;
WHERE
      WR1: (tensor_order=1) OR (tensor_order=2) OR (tensor_order=4);
      WR2: ((tensor_order=1) AND (dimensionality>=2)) OR
      ((tensor_order=2) AND ((dimensionality=2) OR (dimensionality=3))) OR
      ((tensor_order=4) AND ((dimensionality=2) OR (dimensionality=3)));
END_ENTITY;
ENTITY SIR_scalar_in_tensor ;
      tensor: SIR_property_tensor;
      scalar: SIR_property_scalar;
      position: INTEGER;
      roles: SIR_property_meaning;
UNIQUE
      UR1: position, tensor;
WHERE
      WR1: (position>=1) AND (position<=
(tensor.dimensionality**tensor.tensor_order));
END_ENTITY;
ENTITY SIR_property_meaning;
      roles: LIST[1:?] OF UNIQUE type_qualifier;
END_ENTITY;
ENTITY SIR_aspect;
      name: label;
      valued_property: SIR_property_usage;
      lifetime: SIR_lifetime_type;
      component_sequence: SIR_component_sequence;
      scans: LIST[1:?] OF SIR_scan;
WHERE
      WR1: (SIZEOF(scans)=1) OR
(SIZEOF(QUERY(scan<*scans|('NRF_SCHEMA.SIR_SCAN_DERIVED' IN TYPEOF(SELF))))=0)
OR (NOT EXISTS(valued_property.ato_case.abscissa_sequencing));
END_ENTITY;
ENTITY SIR_ato_campaign
SUBTYPE OF (group);
      of_project: organizational_project;
      in_context: application_context_element;
```

```
END_ENTITY;

ENTITY SIR_ato_case;

      id: identifier;
      name: label;
      description: text;
      root_model: SIR_model;
      abscissa_class: SIR_property_usage;
      abscissa_sequencing: OPTIONAL SIR_abscissa_sequencing_type;

INVERSE

      campaign_membership: SIR_case_in_campaign FOR cases;
UNIQUE

      UR1: id, campaign_membership;
WHERE

      WR1: SELF :=: abscissa_class.ato_case;
      WR2: ('NRF_SCHEMA.SIR_PROPERTY_QUANTITATIVE' IN
TYPEOF(abscissa_class.property))
OR ('NRF_SCHEMA.SIR_PROPERTY_DESCRIPTIVE' IN TYPEOF(abscissa_class.property));

END_ENTITY;

ENTITY SIR_case_in_campaign

SUBTYPE OF (group_assignment);

      SELF\group_assignment.assigned_group: SIR_ato_campaign;

      cases: SET[1:?] OF SIR_ato_case;

END_ENTITY;

ENTITY SIR_ato_phase;

      id: identifier;

      name: label;

      description: text;

      of_case: SIR_ato_case;

UNIQUE

      UR1: id, of_case;

END_ENTITY;

ENTITY SIR_initial_conditioning;

      conditioned_data: SIR_case_or_phase;

      initial_condition: SIR_aspect;

END_ENTITY;

ENTITY SIR_property_usage

SUBTYPE OF (variable_semantics);

      of_case: SIR_ato_case;

      property: SIR_property_class;
```

```
      meaning: SIR_property_meaning;

END_ENTITY;

ENTITY SIR_scan;

      abscissa_value: OPTIONAL SIR_number_or_string;

      mask: OPTIONAL SIR_mask;

      values: LIST[1:?] OF SIR_number_or_string;

INVERSE

      aspect: SIR_aspect FOR scans;

END_ENTITY;

ENTITY SIR_mask;

      defined_ranges: SET[1:?] OF SIR_index_interval;

END_ENTITY;

ENTITY SIR_unit_assignment;
      in_case: SIR_ato_case;
      for_property: SIR_property_scalar;
      assigned_unit: unit;
UNIQUE
      UR1: in_case, for_property, assigned_unit;
END_ENTITY;

FUNCTION acyclic_product_definition_relationship
     (relation          : product_definition_relationship;
      relatives         : SET OF product_definition;
      specific_relation : STRING) : LOGICAL;

     LOCAL
       x               : SET OF product_definition_relationship;
       i               : INTEGER;
       local_relatives : SET OF product_definition;
     END_LOCAL;

     REPEAT i := 1 TO HIINDEX(relatives);
       IF relation.relating_product_definition :=: relatives[i] THEN
         RETURN(FALSE);
       END_IF;
     END_REPEAT;
     x := bag_to_set(USEDIN (relation.relating_product_definition,
specific_relation));
          local_relatives := relatives +
relation.relating_product_definition;
     IF SIZEOF(x) > 0 THEN
       REPEAT i := 1 TO HIINDEX(x);
         IF NOT acyclic_product_definition_relationship
                (x[i], local_relatives, specific_relation) THEN
           RETURN(FALSE);
         END_IF;
       END_REPEAT;
     END_IF;
     RETURN(TRUE);
   END_FUNCTION;

FUNCTION valid_time (time: local_time) : BOOLEAN;
     IF EXISTS (time.second_component) THEN
       RETURN (EXISTS (time.minute_component));
     ELSE
       RETURN (TRUE);
```

```
    END_IF;
  END_FUNCTION;

FUNCTION valid_calendar_date (date : calendar_date) : LOGICAL;

    IF NOT ({1 <= date.day_component <= 31}) THEN
      RETURN(FALSE);
    END_IF;
    CASE date.month_component OF
      4         : RETURN({ 1<= date.day_component <= 30});
      6         : RETURN({ 1<= date.day_component <= 30});
      9         : RETURN({ 1<= date.day_component <= 30});
      11        : RETURN({ 1<= date.day_component <= 30});
      2         :
      BEGIN
        IF (leap_year(date.year_component)) THEN
          RETURN({ 1<= date.day_component <= 29});
        ELSE
          RETURN({ 1<= date.day_component <= 28});
        END_IF;
      END;
      OTHERWISE : RETURN(TRUE);
    END_CASE;
  END_FUNCTION;

FUNCTION leap_year(year : year_number) : BOOLEAN;

    IF ((((year MOD 4) = 0) AND ((year MOD 100) <> 0)) OR
        ((year MOD 400) = 0)) THEN
      RETURN(TRUE);
    ELSE
      RETURN(FALSE);
    END_IF;
  END_FUNCTION;

FUNCTION bag_to_set (the_bag : BAG OF GENERIC : intype) : SET OF GENERIC
: intype;

    LOCAL

      the_set: SET OF GENERIC : intype := [];
      i      : INTEGER;
    END_LOCAL;

    IF SIZEOF (the_bag) > 0 THEN
      REPEAT i := 1 to HIINDEX (the_bag);
        the_set := the_set + the_bag [i];
      END_REPEAT;
    END_IF;

    RETURN (the_set);

  END_FUNCTION;

FUNCTION valid_units ( m : measure_with_unit ) : BOOLEAN ;

    IF 'NRF_SCHEMA.LENGTH_MEASURE' IN TYPEOF ( m.value_component )
THEN
      IF derive_dimensional_exponents ( m.unit_component ) <>
         dimensional_exponents ( 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 ) THEN
        RETURN (FALSE);
      END_IF;
    END_IF;

    IF 'NRF_SCHEMA.MASS_MEASURE' IN TYPEOF ( m.value_component ) THEN
      IF derive_dimensional_exponents ( m.unit_component ) <>
         dimensional_exponents ( 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0 ) THEN
        RETURN (FALSE);
```

```
      END_IF;
    END_IF;

    IF 'NRF_SCHEMA.TIME_MEASURE' IN TYPEOF ( m.value_component ) THEN
      IF derive_dimensional_exponents ( m.unit_component ) <>
         dimensional_exponents ( 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0 ) THEN
        RETURN (FALSE);
      END_IF;
    END_IF;

   IF 'NRF_SCHEMA.ELECTRIC_CURRENT_MEASURE' IN TYPEOF (
m.value_component ) THEN
      IF derive_dimensional_exponents ( m.unit_component ) <>
         dimensional_exponents ( 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0 ) THEN
        RETURN (FALSE);
      END_IF;
    END_IF;

    IF 'NRF_SCHEMA.THERMODYNAMIC_TEMPERATURE_MEASURE'
    IN TYPEOF ( m.value_component ) THEN
      IF derive_dimensional_exponents ( m.unit_component ) <>
         dimensional_exponents ( 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0 ) THEN
        RETURN (FALSE);
      END_IF;
    END_IF;

    IF 'NRF_SCHEMA.AMOUNT_OF_SUBSTANCE_MEASURE' IN TYPEOF (
m.value_component ) THEN
      IF derive_dimensional_exponents ( m.unit_component ) <>
         dimensional_exponents ( 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0 ) THEN
        RETURN (FALSE);
      END_IF;
    END_IF;

    IF 'NRF_SCHEMA.LUMINOUS_INTENSITY_MEASURE' IN TYPEOF (
m.value_component ) THEN
      IF derive_dimensional_exponents ( m.unit_component ) <>
         dimensional_exponents ( 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0 ) THEN
        RETURN (FALSE);
      END_IF;
    END_IF;

    IF 'NRF_SCHEMA.PLANE_ANGLE_MEASURE' IN TYPEOF ( m.value_component
) THEN
      IF derive_dimensional_exponents ( m.unit_component ) <>
         dimensional_exponents ( 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 ) THEN
        RETURN (FALSE);
      END_IF;
    END_IF;

    IF 'NRF_SCHEMA.SOLID_ANGLE_MEASURE' IN TYPEOF ( m.value_component
) THEN
      IF derive_dimensional_exponents ( m.unit_component ) <>
         dimensional_exponents ( 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 ) THEN
        RETURN (FALSE);
      END_IF;
    END_IF;

    IF 'NRF_SCHEMA.AREA_MEASURE' IN TYPEOF ( m.value_component ) THEN
      IF derive_dimensional_exponents ( m.unit_component ) <>
         dimensional_exponents ( 2.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 ) THEN
        RETURN (FALSE);
      END_IF;
    END_IF;

    IF 'NRF_SCHEMA.VOLUME_MEASURE' IN TYPEOF ( m.value_component )
THEN
      IF derive_dimensional_exponents ( m.unit_component ) <>
```

```
           dimensional_exponents ( 3.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 ) THEN
          RETURN (FALSE);
        END_IF;
      END_IF;

      IF 'NRF_SCHEMA.RATIO_MEASURE' IN TYPEOF ( m.value_component ) THEN
        IF derive_dimensional_exponents ( m.unit_component ) <>
           dimensional_exponents ( 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 ) THEN
          RETURN (FALSE);
        END_IF;
      END_IF;

      IF 'NRF_SCHEMA.POSITIVE_LENGTH_MEASURE' IN TYPEOF (
m.value_component ) THEN
        IF derive_dimensional_exponents ( m.unit_component ) <>
           dimensional_exponents ( 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 ) THEN
          RETURN (FALSE);
        END_IF;
      END_IF;

      IF 'NRF_SCHEMA.POSITIVE_PLANE_ANGLE_MEASURE' IN TYPEOF (
m.value_component ) THEN
        IF derive_dimensional_exponents ( m.unit_component ) <>
           dimensional_exponents ( 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 ) THEN
          RETURN (FALSE);
        END_IF;
      END_IF;

      RETURN (TRUE);

    END_FUNCTION;

FUNCTION derive_dimensional_exponents (x : unit) :
dimensional_exponents;

      LOCAL
        i      : INTEGER;
        result : dimensional_exponents :=
                 dimensional_exponents(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0);
      END_LOCAL;

      IF 'NRF_SCHEMA.DERIVED_UNIT' IN TYPEOF(x) THEN
        REPEAT i := LOINDEX(x.elements) TO HIINDEX(x.elements);

          result.length_exponent                    :=
            result.length_exponent +
            (x.elements[i].exponent *
             x.elements[i].unit.dimensions.length_exponent);

          result.mass_exponent                      :=
            result.mass_exponent +
            (x.elements[i].exponent *
             x.elements[i].unit.dimensions.mass_exponent);

         result.time_exponent :=
            result.time_exponent +
            (x.elements[i].exponent *
             x.elements[i].unit.dimensions.time_exponent);

          result.electric_current_exponent          :=
            result.electric_current_exponent +
            (x.elements[i].exponent *
             x.elements[i].unit.dimensions.electric_current_exponent);

          result.thermodynamic_temperature_exponent :=
            result.thermodynamic_temperature_exponent +
            (x.elements[i].exponent *
```

```
x.elements[i].unit.dimensions.thermodynamic_temperature_exponent);

         result.amount_of_substance_exponent       :=
           result.amount_of_substance_exponent +
           (x.elements[i].exponent *
            x.elements[i].unit.dimensions.amount_of_substance_exponent);

         result.luminous_intensity_exponent        :=
           result.luminous_intensity_exponent +
           (x.elements[i].exponent *
            x.elements[i].unit.dimensions.luminous_intensity_exponent);

       END_REPEAT;
     ELSE
       result := x.dimensions;
     END_IF;
     RETURN (result);
   END_FUNCTION;

FUNCTION dimensions_for_si_unit (n : si_unit_name) :
dimensional_exponents;

     CASE n  OF
       metre          : RETURN (dimensional_exponents
                                (1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0));
       gram           : RETURN (dimensional_exponents
                                (0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0));
       second         : RETURN (dimensional_exponents
                                (0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0));
       ampere         : RETURN (dimensional_exponents
                                (0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0));
       kelvin         : RETURN (dimensional_exponents
                                (0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0));
       mole           : RETURN (dimensional_exponents
                                (0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0));
       candela        : RETURN (dimensional_exponents
                                (0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0));
       radian         : RETURN (dimensional_exponents
                                (0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0));
       steradian      : RETURN (dimensional_exponents
                                (0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0));
       hertz          : RETURN (dimensional_exponents
                                (0.0, 0.0, -1.0, 0.0, 0.0, 0.0, 0.0));
       newton         : RETURN (dimensional_exponents
                                (1.0, 1.0, -2.0, 0.0, 0.0, 0.0, 0.0));
       pascal         : RETURN (dimensional_exponents
                                (-1.0, 1.0, -2.0, 0.0, 0.0, 0.0, 0.0));
       joule          : RETURN (dimensional_exponents
                                (2.0, 1.0, -2.0, 0.0, 0.0, 0.0, 0.0));
       watt           : RETURN (dimensional_exponents
                                (2.0, 1.0, -3.0, 0.0, 0.0, 0.0, 0.0));
       coulomb        : RETURN (dimensional_exponents
                                (0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0));
       volt           : RETURN (dimensional_exponents
                                (2.0, 1.0, -3.0, -1.0, 0.0, 0.0, 0.0));
       farad          : RETURN (dimensional_exponents
                                (-2.0, -1.0, 4.0, 1.0, 0.0, 0.0, 0.0));
       ohm            : RETURN (dimensional_exponents
                                (2.0, 1.0, -3.0, -2.0, 0.0, 0.0, 0.0));
       siemens        : RETURN (dimensional_exponents
                                (-2.0, -1.0, 3.0, 2.0, 0.0, 0.0, 0.0));
       weber          : RETURN (dimensional_exponents
                                (2.0, 1.0, -2.0, -1.0, 0.0, 0.0, 0.0));
       tesla          : RETURN (dimensional_exponents
                                (0.0, 1.0, -2.0, -1.0, 0.0, 0.0, 0.0));
       henry          : RETURN (dimensional_exponents
                                (2.0, 1.0, -2.0, -2.0, 0.0, 0.0, 0.0));
       degree_Celsius : RETURN (dimensional_exponents
```

```
                                    (0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0));
        lumen          : RETURN (dimensional_exponents
                                    (0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0));
        lux            : RETURN (dimensional_exponents
                                    (-2.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0));
        becquerel      : RETURN (dimensional_exponents
                                    (0.0, 0.0, -1.0, 0.0, 0.0, 0.0, 0.0));
        gray           : RETURN (dimensional_exponents
                                    (2.0, 0.0, -2.0, 0.0, 0.0, 0.0, 0.0));
        sievert        : RETURN (dimensional_exponents
                                    (2.0, 0.0, -2.0, 0.0, 0.0, 0.0, 0.0));
     END_CASE;
   END_FUNCTION;

FUNCTION using_representations (item : representation_item)
  : SET OF representation;
  LOCAL
    results            : SET OF representation;
    result_bag         : BAG OF representation;
    intermediate_items : SET OF representation_item;
    i                  : INTEGER;
  END_LOCAL;

  result_bag := USEDIN(item,'NRF_SCHEMA.REPRESENTATION.ITEMS');

  IF SIZEOF(result_bag) > 0 THEN
    REPEAT i := 1 TO HIINDEX(result_bag);
      results := results + result_bag[i];
    END_REPEAT;
  END_IF;


  intermediate_items := QUERY(z <* bag_to_set( USEDIN(item , '')) |
    'NRF_SCHEMA.REPRESENTATION_ITEM' IN TYPEOF(z));

  IF SIZEOF(intermediate_items) > 0 THEN

    REPEAT i := 1 TO HIINDEX(intermediate_items);
      results := results + using_representations(intermediate_items[i]);
    END_REPEAT;
  END_IF;

  RETURN (results);
END_FUNCTION;

FUNCTION item_in_context
  (item  : representation_item;
   cntxt : representation_context) : BOOLEAN;
  LOCAL
    i : INTEGER;
    y : BAG OF representation_item;
  END_LOCAL;

  IF SIZEOF(USEDIN(item,'NRF_SCHEMA.REPRESENTATION.ITEMS')
    * cntxt.representations_in_context) > 0 THEN
    RETURN (TRUE);

    ELSE
    y := QUERY(z <* USEDIN (item , '') |
         'NRF_SCHEMA.REPRESENTATION_ITEM' IN TYPEOF(z));

      IF SIZEOF(y) > 0 THEN

      REPEAT i := 1 TO HIINDEX(y);

        IF item_in_context(y[i], cntxt) THEN
          RETURN (TRUE);
        END_IF;
```

```
      END_REPEAT;
    END_IF;
  END_IF;
  RETURN (FALSE);
END_FUNCTION;

FUNCTION acyclic_mapped_representation
  (parent_set   : SET OF representation;
   children_set : SET OF representation_item) : BOOLEAN;
  LOCAL
    x,y : SET OF representation_item;
    i   : INTEGER;
  END_LOCAL;

  x := QUERY(z <* children_set | 'NRF_SCHEMA.MAPPED_ITEM'
       IN TYPEOF(z));

  IF SIZEOF(x) > 0 THEN

    REPEAT i := 1 TO HIINDEX(x);

      IF x[i]\mapped_item.mapping_source.mapped_representation
        IN parent_set THEN
        RETURN (FALSE);
      END_IF;
      IF NOT acyclic_mapped_representation
        (parent_set +
x[i]\mapped_item.mapping_source.mapped_representation,
        x[i]\mapped_item.mapping_source.mapped_representation.items) THEN
        RETURN (FALSE);
      END_IF;
    END_REPEAT;
  END_IF;

  x := children_set - x;

  IF SIZEOF(x) > 0 THEN

    REPEAT i := 1 TO HIINDEX(x);


      y := QUERY(z <* bag_to_set( USEDIN(x[i], '')) |
           'NRF_SCHEMA.REPRESENTATION_ITEM' IN TYPEOF(z));

      IF NOT acyclic_mapped_representation(parent_set, y) THEN
        RETURN (FALSE);
      END_IF;
    END_REPEAT;
  END_IF;
  RETURN (TRUE);
END_FUNCTION;

FUNCTION acyclic (arg1 : generic_expression;  arg2 : SET OF generic_expression):
BOOLEAN;
     LOCAL
          result: BOOLEAN;
     END_LOCAL;
     IF ('NRF_SCHEMA.SIMPLE_GEN_EXPRESSION'
          IN TYPEOF (arg1))
     THEN RETURN (TRUE);
     END_IF;

     IF arg1 IN arg2
     THEN RETURN (FALSE);
     END_IF;

     IF 'NRF_SCHEMA.UNARY_GEN_EXPRESSION'
          IN TYPEOF (arg1)
```

```
      THEN RETURN (acyclic(arg1\unary_gen_expression.operand,arg2+[arg1]));
      END_IF;

      IF 'NRF_SCHEMA.BINARY_GEN_EXPRESSION'
            IN TYPEOF (arg1)
      THEN RETURN (acyclic(arg1\binary_gen_expression.operands[1],
                  arg2+[arg1]) AND acyclic(arg1\binary_gen_expression.operands[2],
                  arg2+[arg1]));
      END_IF;

      IF 'NRF_SCHEMA.M_ARY_GEN_EXPRESSION'
            IN TYPEOF (arg1)
      THEN result := TRUE;
      REPEAT i := 1 TO SIZEOF (arg1\m_ary_gen_expression.operands);
            result := result AND
            acyclic(arg1\m_ary_gen_expression.operands[i], arg2+[arg1]);
      END_REPEAT;
      RETURN (result);
      END_IF;
END_FUNCTION;

FUNCTION is_acyclic (arg : generic_expression): BOOLEAN;
      RETURN (acyclic (arg, []));
END_FUNCTION ;

FUNCTION syntactic_variable_type (arg : generic_expression): STRING;
IF 'NRF_SCHEMA.INT_NUMERIC_VARIABLE' IN TYPEOF (arg)
THEN
      RETURN ('INTEGER');
END_IF;
IF 'NRF_SCHEMA.REAL_NUMERIC_VARIABLE'IN TYPEOF (arg)
THEN
      RETURN ('REAL');
END_IF;
IF 'NRF_SCHEMA.BOOLEAN_VARIABLE' IN TYPEOF (arg)
THEN
      RETURN ('BOOLEAN');
END_IF;
IF 'NRF_SCHEMA_STRING_VARIABLE' IN TYPEOF (arg)
THEN
      RETURN ('STRING');
END_IF;
RETURN ('ERROR_TYPE');
END_FUNCTION;

FUNCTION syntactic_boolean_expression_type (arg : generic_expression):
                        STRING;
LOCAL
      c_arg : comparison_expression ;
      i : INTEGER ;
END_LOCAL ;

IF 'NRF_SCHEMA.BOOLEAN_LITERAL' IN TYPEOF (arg)
THEN
      RETURN ('BOOLEAN');
END_IF;
IF 'NRF_SCHEMA.VARIABLE' IN TYPEOF (arg)
THEN
      RETURN (syntactic_variable_type (arg)) ;
END_IF;


IF 'NRF_SCHEMA.COMPARISON_EXPRESSION' IN TYPEOF (arg)
THEN
      c_arg := arg\comparison_expression ;

IF 'NRF_SCHEMA.COMPARISON_EQUAL' IN TYPEOF (c_arg)
THEN
```

```
      IF (syntactic_expression_type (c_arg.operands [1]) ='STRING')
                  AND (syntactic_expression_type (c_arg.operands [2]) ='STRING')
      THEN
            RETURN ('BOOLEAN');
      END_IF;
      IF ((syntactic_expression_type(c_arg.operands[1]) = 'INTEGER')
                  OR (syntactic_expression_type (c_arg.operands [1]) = 'REAL')
                  AND ( syntactic_expression_type(c_arg.operands [2]) =
                  'INTEGER')
                  OR (syntactic_expression_type (c_arg.operands [2]) = 'REAL'))
      THEN
            RETURN ('BOOLEAN');
      ELSE
            RETURN ('ERROR_TYPE') ;
      END_IF;
END_IF;

IF 'NRF_SCHEMA.COMPARISON_GREATER' IN TYPEOF (c_arg)
THEN
      IF (syntactic_expression_type (c_arg.operands [1]) = 'STRING')
                  AND (syntactic_expression_type (c_arg.operands [2]) ='STRING')
      THEN
            RETURN ('BOOLEAN');
      END_IF;
      IF ((syntactic_expression_type (c_arg.operands[1]) ='INTEGER')
                  OR (syntactic_expression_type (c_arg.operands [1]) = 'REAL'))
                  AND ((syntactic_expression_type (c_arg.operands [2]) = 'INTEGER')
                  OR (syntactic_expression_type (c_arg.operands [2]) = 'REAL'))
      THEN
            RETURN ('BOOLEAN');
      ELSE
            RETURN ('ERROR_TYPE') ;
      END_IF;
END_IF;

IF'NRF_SCHEMA.COMPARISON_GREATER_EQUAL'IN
TYPEOF(c_arg)
THEN
      IF (syntactic_expression_type (c_arg.operands [1]) = 'STRING')
                  AND (syntactic_expression_type (c_arg.operands [2]) = 'STRING')
      THEN
            RETURN ('BOOLEAN');
      END_IF;
      IF ((syntactic_expression_type (c_arg.operands[1]) = 'INTEGER')
                  OR (syntactic_expression_type (c_arg.operands [1]) = 'REAL'))
                  AND ((syntactic_expression_type (c_arg.operands [2]) = 'INTEGER')
                  OR (syntactic_expression_type (c_arg.operands [2]) = 'REAL'))
      THEN
            RETURN ('BOOLEAN');
      ELSE
            RETURN ('ERROR_TYPE');
      END_IF;
END_IF;

IF 'NRF_SCHEMA.COMPARISON_LESS' IN TYPEOF (c_arg)
THEN
      IF (syntactic_expression_type (c_arg.operands [1]) = 'STRING')
                  AND (syntactic_expression_type (c_arg.operands [2]) = 'STRING')
      THEN
            RETURN ('BOOLEAN');
      END_IF;
      IF ((syntactic_expression_type (c_arg.operands[1]) = 'INTEGER')
                  OR (syntactic_expression_type (c_arg.operands [1]) = 'REAL'))
                  AND ((syntactic_expression_type (c_arg.operands [2]) = 'INTEGER')
                  OR (syntactic_expression_type (c_arg.operands [2]) = 'REAL'))
      THEN
            RETURN ('BOOLEAN');
      ELSE
```

```
            RETURN ('ERROR_TYPE') ;
      END_IF;
END_IF;

IF 'NRF_SCHEMA.COMPARISON_LESS_EQUAL' IN TYPEOF
(c_arg)
THEN
      IF (syntactic_expression_type (c_arg.operands [1]) = 'STRING')
                  AND (syntactic_expression_type (c_arg.operands [2]) = 'STRING')
      THEN
            RETURN ('BOOLEAN');
      END_IF;
      IF ((syntactic_expression_type (c_arg.operands[1]) = 'INTEGER')
                  OR (syntactic_expression_type (c_arg.operands [1]) = 'REAL'))
                  AND ((syntactic_expression_type (c_arg.operands [2]) = 'INTEGER')
                  OR (syntactic_expression_type (c_arg.operands [2]) = 'REAL'))
      THEN
            RETURN ('BOOLEAN');
      ELSE
            RETURN ('ERROR_TYPE');
      END_IF;
END_IF;

IF 'NRF_SCHEMA.COMPARISON_NOT_EQUAL' IN TYPEOF
(c_arg)
THEN
      IF ((syntactic_expression_type (c_arg.operands [1]) = 'STRING')
                  AND (syntactic_expression_type (c_arg.operands [2]) = 'STRING'))
      THEN
            RETURN ('BOOLEAN');
      END_IF;
      IF ((syntactic_expression_type (c_arg.operands[1]) = 'INTEGER')
                  OR (syntactic_expression_type (c_arg.operands [1]) = 'REAL'))
                  AND ((syntactic_expression_type (c_arg.operands [2]) = 'INTEGER')
                  OR (syntactic_expression_type (c_arg.operands [2]) = 'REAL'))
      THEN
            RETURN ('BOOLEAN');
      ELSE
            RETURN ('ERROR_TYPE');
      END_IF;
END_IF;

IF 'NRF_SCHEMA.LIKE_EXPRESSION' IN TYPEOF (c_arg)
THEN
      IF ((syntactic_expression_type (c_arg.operands [1]) = 'STRING')
                  AND (syntactic_expression_type (c_arg.operands [2]) = 'STRING'))
      THEN
            RETURN ('BOOLEAN');
      ELSE
            RETURN ('ERROR_TYPE');
      END_IF;
END_IF;
END_IF;

END_FUNCTION;

FUNCTION syntactic_numeric_expression_type (arg : generic_expression) : STRING;
LOCAL
      i : INTEGER ;
      u_arg : unary_n_expression ;
      m_arg : m_ary_n_expression ;
      b_arg : binary_n_expression ;
      bool : BOOLEAN ;
END_LOCAL ;
IF 'NRF_SCHEMA.INT_LITERAL' IN TYPEOF (arg)
      THEN RETURN ('INTEGER');
END_IF;
IF 'NRF_SCHEMA.REAL_LITERAL' IN TYPEOF (arg)
```

```
     THEN RETURN ('REAL');
END_IF;
IF 'NRF_SCHEMA.VARIABLE' IN TYPEOF (arg)
     THEN RETURN (syntactic_variable_type (arg));
END_IF;

IF 'NRF_SCHEMA.BINARY_N_EXPRESSION' IN TYPEOF (arg)
       THEN b_arg := arg\binary_n_expression;

IF 'NRF_SCHEMA.MINUS_EXPRESSION' IN TYPEOF (b_arg)
THEN
     IF (syntactic_numeric_expression_type(b_arg.operands [1]) =
'INTEGER')
            AND (syntactic_numeric_expression_type(
            b_arg.operands [2]) = 'INTEGER')
     THEN
          RETURN ('INTEGER');
     ELSE
          IF ((syntactic_numeric_expression_type(b_arg.operands [1]) =
'REAL')
                OR (syntactic_numeric_expression_type(
                      b_arg.operands [1]) = 'INTEGER'))
                AND (syntactic_numeric_expression_type(
                      b_arg.operands [2]) = 'REAL')
                OR (syntactic_numeric_expression_type(
                      b_arg.operands [2]) = 'INTEGER')
          THEN
               RETURN ('REAL') ;
          ELSE
               RETURN ('ERROR_TYPE') ;
          END_IF;
     END_IF;
END_IF;

IF 'NRF_SCHEMA.DIV_EXPRESSION' IN TYPEOF (b_arg)
THEN
     IF (syntactic_numeric_expression_type(b_arg.operands [1]) =
'INTEGER')
            AND (syntactic_numeric_expression_type(
                     b_arg.operands [2]) = 'INTEGER')
     THEN
          RETURN ('INTEGER');
     ELSE
          RETURN ('ERROR_TYPE');
     END_IF;
END_IF;

IF 'NRF_SCHEMA.MOD_EXPRESSION' IN TYPEOF (b_arg)
THEN
     IF (syntactic_numeric_expression_type(b_arg.operands [1]) =
'INTEGER')
             AND (syntactic_numeric_expression_type(
                       b_arg.operands [2]) = 'INTEGER')
     THEN
          RETURN ('INTEGER');
     ELSE
          RETURN ('ERROR_TYPE');
     END_IF;
END_IF;

IF'NRF_SCHEMA.SLASH_EXPRESSION' IN TYPEOF (b_arg)
THEN
     IF ((syntactic_numeric_expression_type(b_arg.operands [1]) = 'REAL')
            OR (syntactic_numeric_expression_type(
                  b_arg.operands [1]) = 'INTEGER'))
            AND (syntactic_numeric_expression_type(
                     b_arg.operands [2]) = 'REAL')
            OR (syntactic_numeric_expression_type(
```

```
                        b_arg.operands [2]) = 'INTEGER')
      THEN
            RETURN ('REAL') ;
      ELSE
            RETURN ('ERROR_TYPE') ;
      END_IF;
END_IF;

IF 'NRF_SCHEMA.POWER_EXPRESSION' IN TYPEOF (b_arg)
THEN
      IF (syntactic_numeric_expression_type(b_arg.operands [1]) =
'INTEGER')
      THEN
            IF (syntactic_numeric_expression_type(b_arg.operands[2]) =
'INTEGER')
            THEN
                  RETURN ('INTEGER') ;
            END_IF;
            IF (syntactic_numeric_expression_type(b_arg.operands [2]) = 'REAL')
            THEN
                  RETURN ('REAL');
            ELSE
                  RETURN ('ERROR_TYPE');
            END_IF;
      END_IF;
      IF (syntactic_numeric_expression_type(b_arg.operands [1]) = 'REAL')
      THEN
            IF (syntactic_numeric_expression_type(b_arg.operands[2]) =
'INTEGER')
                        OR (syntactic_numeric_expression_type(b_arg.operands[2])=
'REAL')
            THEN
                  RETURN ('REAL');
            ELSE
                  RETURN ('ERROR_TYPE');
            END_IF;
      END_IF;
END_IF;
RETURN ('ERROR_TYPE');

END_IF;

IF 'NRF_SCHEMA.M_ARY_N_EXPRESSION' IN TYPEOF (arg)
THEN
      m_arg := arg\m_ary_n_expression ;

IF 'NRF_SCHEMA.PLUS_EXPRESSION' IN TYPEOF (m_arg)
THEN bool := TRUE;
REPEAT i := 1 TO SIZEOF(m_arg.operands);
      IF (syntactic_numeric_expression_type(m_arg.operands[i])
                  <> 'INTEGER')
      THEN bool:= FALSE ;
      END_IF;
END_REPEAT;
      IF bool THEN RETURN ('INTEGER');
      END_IF;
      bool := TRUE;
      REPEAT i := 1 TO SIZEOF(m_arg.operands);
            IF (syntactic_numeric_expression_type(m_arg.operands[i])
                  <> 'REAL' )
                  AND (syntactic_numeric_expression_type(m_arg.operands[i])
                  <> 'INTEGER' )
            THEN bool:= FALSE ;
            END_IF;
      END_REPEAT;
      IF bool THEN RETURN ('REAL');
      ELSE RETURN ('ERROR_TYPE');
      END_IF;
```

```
END_IF;

IF 'NRF_SCHEMA.MULT_EXPRESSION' IN TYPEOF (m_arg)
THEN bool := TRUE;
REPEAT i := 1 TO SIZEOF(m_arg.operands);
      IF (syntactic_numeric_expression_type(m_arg.operands[i])
                  <> 'INTEGER')
      THEN bool:= FALSE ;
      END_IF;
END_REPEAT;
      IF bool THEN RETURN ('INTEGER');
      END_IF;
      bool := TRUE;
      REPEAT i := 1 TO SIZEOF(m_arg.operands);
            IF (syntactic_numeric_expression_type(m_arg.operands[i]) <> 'REAL'
)
                  AND (syntactic_numeric_expression_type(m_arg.operands[i])
                  <> 'INTEGER' )
            THEN bool:= FALSE ;
            END_IF;
      END_REPEAT;
      IF bool
      THEN
            RETURN ('REAL');
      ELSE
            RETURN ('ERROR_TYPE');
      END_IF;
END_IF;

IF 'NRF_SCHEMA.MAXIMUM_FUNCTION' IN TYPEOF (m_arg)
THEN bool := TRUE;
REPEAT i := 1 TO SIZEOF(m_arg.operands);
      IF (syntactic_numeric_expression_type(m_arg.operands[i])
                  <> 'INTEGER')
      THEN bool:= FALSE ;
      END_IF;
END_REPEAT;
      IF bool
      THEN
            RETURN ('INTEGER');
      END_IF;
      bool := TRUE;
      REPEAT i := 1 TO SIZEOF(m_arg.operands);
            IF (syntactic_numeric_expression_type(m_arg.operands[i]) <> 'REAL'
)
                        AND (syntactic_numeric_expression_type(m_arg.operands[i])
                        <> 'INTEGER' )
            THEN bool:= FALSE ;
            END_IF;
      END_REPEAT;
      IF bool
      THEN
            RETURN ('REAL');
      ELSE
            RETURN ('ERROR_TYPE');
      END_IF;
END_IF;

IF 'NRF_SCHEMA.MINIMUM_FUNCTION' IN TYPEOF (m_arg)
THEN bool := TRUE;
REPEAT i := 1 TO SIZEOF(m_arg.operands);
      IF    (syntactic_numeric_expression_type(m_arg.operands[i]) <>
'INTEGER')
      THEN bool:= FALSE ;
      END_IF;
END_REPEAT;
      IF bool THEN RETURN ('INTEGER');
      END_IF;
```

```
      bool := TRUE;
      REPEAT i := 1 TOSIZEOF(m_arg.operands);
            IF (syntactic_numeric_expression_type(m_arg.operands[i]) <> 'REAL'
)
                        AND (syntactic_numeric_expression_type(m_arg.operands[i])
                        <> 'INTEGER' )
            THEN bool:= FALSE ;
            END_IF;
      END_REPEAT;
      IF bool
      THEN
            RETURN ('REAL');
      ELSE
            RETURN ('ERROR_TYPE');
      END_IF;
END_IF;

IF 'NRF_SCHEMA.REAL_DEFINED_FUNCTION' IN TYPEOF(arg)

THEN
      RETURN('REAL');
END_IF;
IF 'NRF_SCHEMA.INTEGER_DEFINED_FUNCTION' IN
TYPEOF(arg)
THEN
      RETURN('INTEGER');
END_IF;
RETURN ('ERROR_TYPE');
END_IF;

END_FUNCTION;

FUNCTION syntactic_expression_type (arg : generic_expression) : STRING;
IF 'NRF_SCHEMA.NUMERIC_EXPRESSION' IN TYPEOF (arg)
THEN
      RETURN (syntactic_numeric_expression_type (arg)) ;
END_IF;
IF 'NRF_SCHEMA.BOOLEAN_EXPRESSION' IN TYPEOF (arg)
THEN
      RETURN (syntactic_boolean_expression_type (arg));
END_IF;
END_FUNCTION;

FUNCTION is_SQL_mappable (arg: generic_expression) : BOOLEAN;
      LOCAL
            i: INTEGER;
      END_LOCAL;
      IF    'NRF_SCHEMA.SIMPLE_N_EXPRESSION'
                  IN TYPEOF (arg)
      THEN
            RETURN (TRUE);
      END_IF;
      IF    'NRF_SCHEMA.SQL_MAPPABLE_DEFINED_FUNCTION'
                  IN TYPEOF (arg)
      THEN
            RETURN (TRUE);
      END_IF;
      IF    'NRF_SCHEMA.MINUS_FUNCTION' IN TYPEOF(arg)
      THEN
            RETURN (is_SQL_mappable(arg\unary_n_expression.operand));
      END_IF;
      IF    ('NRF_SCHEMA.ABS_FUNCTION' IN TYPEOF(arg))
                  OR ('NRF_SCHEMA.SIN_FUNCTION' IN TYPEOF(arg))
                  OR ('NRF_SCHEMA.COS_FUNCTION' IN TYPEOF(arg))
                  OR ('NRF_SCHEMA.TAN_FUNCTION' IN TYPEOF(arg))
                  OR ('NRF_SCHEMA.ASIN_FUNCTION' IN TYPEOF(arg))
                  OR ('NRF_SCHEMA.ACOS_FUNCTION' IN TYPEOF(arg))
                  OR ('NRF_SCHEMA.ATAN_FUNCTION' IN TYPEOF(arg))
```

```
                  OR ('NRF_SCHEMA.EXP_FUNCTION' IN TYPEOF(arg))
                  OR ('NRF_SCHEMA.LOG_FUNCTION' IN TYPEOF(arg))
                  OR ('NRF_SCHEMA.LOG2_FUNCTION' IN TYPEOF(arg))
                  OR ('NRF_SCHEMA.LOG10_FUNCTION' IN TYPEOF(arg))
                  OR ('NRF_SCHEMA.SQUARE_ROOT_FUNCTION'
                  IN TYPEOF(arg))
                  OR ('NRF_SCHEMA.VALUE_FUNCTION' IN TYPEOF(arg))
                  OR ('NRF_SCHEMA.LENGTH_FUNCTION'
                       IN TYPEOF(arg))
      THEN
            RETURN (FALSE);
      END_IF;
      IF    ('NRF_SCHEMA.PLUS_EXPRESSION' IN TYPEOF(arg))
                  OR ('NRF_SCHEMA.MULT_EXPRESSION' IN
TYPEOF(arg))
                  OR ('NRF_SCHEMA.MAXIMUM_FUNCTION' IN
TYPEOF(arg))
                  OR ('NRF_SCHEMA.MINIMUM_FUNCTION' IN
TYPEOF(arg))
      THEN
            REPEAT i :=1 TO SIZEOF (arg\m_ary_n_expression.operands);
                  IF NOT is_SQL_mappable(arg\m_ary_n_expression.operands[i])
                  THEN
                        RETURN (FALSE);
                  END_IF;
            END_REPEAT;
            RETURN (TRUE);
      END_IF;
      IF    ('NRF_SCHEMA.MINUS_EXPRESSION' IN TYPEOF(arg))
                  OR ('NRF_SCHEMA.SLASH_EXPRESSION' IN
TYPEOF(arg))
      THEN
            RETURN (is_SQL_mappable(arg\binary_n_expression.operands[1])
            AND is_SQL_mappable(arg\binary_n_expression.operands[2]));
      END_IF;
      IF    ('NRF_SCHEMA.DIV_EXPRESSION' IN TYPEOF(arg))
                  OR ('NRF_SCHEMA.MOD_EXPRESSION' IN TYPEOF(arg))
                  OR ('NRF_SCHEMA.POWER_EXPRESSION' IN
TYPEOF(arg))
      THEN
            RETURN (FALSE);
      END_IF;
      IF 'NRF_SCHEMA.SIMPLE_B_EXPRESSION' IN TYPEOF (arg)
      THEN
            RETURN (TRUE);
      END_IF;
      IF 'NRF_SCHEMA.NOT_EXPRESSION' IN TYPEOF (arg)
      THEN
            RETURN (is_SQL_mappable (arg\UNARY_GEN_EXPRESSION.OPERAND));
      END_IF;
      IF    ('NRF_SCHEMA.ODD_FUNCTION'IN TYPEOF (arg))
                  OR ('NRF_SCHEMA.XOR_EXPRESSION' IN TYPEOF
(arg))
      THEN
            RETURN (FALSE);
      END_IF;
      IF    ('NRF_SCHEMA.AND_EXPRESSION' IN TYPEOF (arg))
                  OR ('NRF_SCHEMA.OR_EXPRESSION' IN TYPEOF (arg))
      THEN
            REPEAT i:=1 TO SIZEOF (arg\M_ARY_GEN_EXPRESSION.OPERANDS);
                  IF NOT is_SQL_mappable (arg\M_ARY_GEN_EXPRESSION.OPERANDS[i])
      THEN
            RETURN (FALSE);
                  END_IF;
            END_REPEAT;
            RETURN (TRUE);
      END_IF;
      IF    'NRF_SCHEMA.EQUALS_EXPRESSION' IN TYPEOF (arg)
```

```
      THEN
            RETURN (is_SQL_mappable (arg\BINARY_GEN_EXPRESSION.OPERANDS [1])
                  AND is_SQL_mappable (arg\BINARY_GEN_EXPRESSION.OPERANDS [2]));
      END_IF;
      IF    ('NRF_SCHEMA.COMPARISON_EQUAL' IN TYPEOF (arg))
                  OR ('NRF_SCHEMA.COMPARISON_GREATER'
                  IN TYPEOF (arg))
                  OR ('NRF_SCHEMA.COMPARISON_GREATER_EQUAL'
                  IN TYPEOF (arg))
                  OR ('NRF_SCHEMA.COMPARISON_LESS' IN TYPEOF
(arg))
                  OR ('NRF_SCHEMA.COMPARISON_LESS_EQUAL'
                  IN TYPEOF (arg))
                  OR ('NRF_SCHEMA.COMPARISON_NOT_EQUAL'
                  IN TYPEOF (arg))
                  OR ('NRF_SCHEMA.LIKE_EXPRESSION' IN TYPEOF
(arg))
      THEN
            RETURN (is_SQL_mappable (arg\COMPARISON_EXPRESSION.OPERANDS[1])
                  AND is_SQL_mappable (arg\COMPARISON_EXPRESSION.OPERANDS[2]));
      END_IF;
      RETURN (FALSE);
END_FUNCTION;

FUNCTION is_int_expr (arg: generic_expression) : BOOLEAN;
      LOCAL
            i: INTEGER;
      END_LOCAL;
      IF 'NRF_SCHEMA.INT_LITERAL' IN TYPEOF(arg)
      THEN
            RETURN (TRUE);
      END_IF;
      IF 'NRF_SCHEMA.REAL_LITERAL' IN TYPEOF(arg)
      THEN
            RETURN (FALSE);
      END_IF;
      IF 'NRF_SCHEMA.INT_NUMERIC_VARIABLE' IN TYPEOF(arg)
      THEN
            RETURN (TRUE);
      END_IF;
      IF 'NRF_SCHEMA.REAL_NUMERIC_VARIABLE' IN
TYPEOF(arg)
      THEN
            RETURN (FALSE);
      END_IF;
      IF 'NRF_SCHEMA.ABS_FUNCTION' IN TYPEOF(arg)
      THEN
            RETURN (is_int_expr(arg\unary_n_expression.operand));
      END_IF;
      IF 'NRF_SCHEMA.MINUS_FUNCTION' IN TYPEOF(arg)
      THEN
            RETURN (is_int_expr(arg\unary_n_expression.operand));
      END_IF;
      IF    ('NRF_SCHEMA.SIN_FUNCTION' IN TYPEOF(arg)) OR
                  ('NRF_SCHEMA.COS_FUNCTION' IN TYPEOF(arg)) OR
                  ('NRF_SCHEMA.TAN_FUNCTION' IN TYPEOF(arg)) OR
                  ('NRF_SCHEMA.ASIN_FUNCTION' IN TYPEOF(arg)) OR
                  ('NRF_SCHEMA.ACOS_FUNCTION' IN TYPEOF(arg)) OR
                  ('NRF_SCHEMA.ATAN_FUNCTION' IN TYPEOF(arg)) OR
                  ('NRF_SCHEMA.EXP_FUNCTION' IN TYPEOF(arg)) OR
                  ('NRF_SCHEMA.LOG_FUNCTION' IN TYPEOF(arg)) OR
                  ('NRF_SCHEMA.LOG2_FUNCTION' IN TYPEOF(arg)) OR
                  ('NRF_SCHEMA.LOG10_FUNCTION' IN TYPEOF(arg)) OR
                  ('NRF_SCHEMA.SQUARE_ROOT_FUNCTION'
                  IN TYPEOF(arg))
      THEN
            RETURN (FALSE);
      END_IF;
```

```
      IF    ('NRF_SCHEMA.PLUS_EXPRESSION' IN TYPEOF(arg))
OR
            ('NRF_SCHEMA.MULT_EXPRESSION' IN TYPEOF(arg)) OR
            ('NRF_SCHEMA.MAXIMUM_FUNCTION' IN TYPEOF(arg)) OR
            ('NRF_SCHEMA.MINIMUM_FUNCTION' IN TYPEOF(arg))
      THEN
            REPEAT i :=1 TO SIZEOF (arg\m_ary_n_expression.operands);
                  IF NOT is_int_expr(arg\m_ary_n_expression.operands[i])
                  THEN
                        RETURN (FALSE);
                  END_IF;
            END_REPEAT;
            RETURN (TRUE);
      END_IF;
      IF    ('NRF_SCHEMA.MINUS_EXPRESSION' IN TYPEOF(arg))
OR
                  ('NRF_SCHEMA.POWER_EXPRESSION' IN TYPEOF(arg))
      THEN
            RETURN (is_int_expr(arg\binary_n_expression.operands[1])
                   AND is_int_expr(arg\binary_n_expression.operands[2]));
      END_IF;
      IF    ('NRF_SCHEMA.DIV_EXPRESSION' IN TYPEOF(arg)) OR
              ('NRF_SCHEMA.MOD_EXPRESSION' IN TYPEOF(arg))
      THEN
            RETURN (TRUE);
      END_IF;
      IF    'NRF_SCHEMA.SLASH_EXPRESSION' IN TYPEOF(arg)
      THEN
            RETURN (FALSE);
      END_IF;
      IF    'NRF_SCHEMA.LENGTH_FUNCTION' IN TYPEOF(arg)
      THEN
            RETURN (TRUE);
      END_IF;
      IF    'NRF_SCHEMA.VALUE_FUNCTION' IN TYPEOF(arg)
      THEN
            IF    'NRF_SCHEMA.INT_VALUE_FUNCTION'
                        IN TYPEOF(arg)
            THEN
                  RETURN (TRUE);
            ELSE
                  RETURN (FALSE);
            END_IF;
      END_IF;
      IF    'NRF_SCHEMA.INTEGER_DEFINED_FUNCTION'
            IN TYPEOF(arg)
      THEN
            RETURN(TRUE) ;
      END_IF ;
      IF    'NRF_SCHEMA.REAL_DEFINED_FUNCTION' IN
TYPEOF(arg) THEN
            RETURN(FALSE) ;
      END_IF ;
      IF    'NRF_SCHEMA.BOOLEAN_DEFINED_FUNCTION'
            IN TYPEOF(arg)
      THEN
            RETURN(FALSE) ;
      END_IF ;
      IF    'NRF_SCHEMA.STRING_DEFINED_FUNCTION'
            IN TYPEOF(arg)
      THEN
            RETURN (FALSE) ;
      END_IF ;

      RETURN (FALSE);
END_FUNCTION;

FUNCTION consistent_size(parameters: LIST[1:?] OF list_ascending_real_literals;
```

```
function_values: list_real_literals):BOOLEAN;

LOCAL
      result: BOOLEAN;
      n_values: INTEGER;
      i: INTEGER;

END_LOCAL;

n_values := SIZEOF(parameters[1]);

REPEAT i:=2 TO SIZEOF(parameters);

      n_values := n_values * SIZEOF(parameters[i]);

END_REPEAT;

result := (SIZEOF(function_values) = n_values);

RETURN(result);

END_FUNCTION;

FUNCTION valid_bounded_variables(bounds:SET OF less_or_greater; variables: LIST OF
SIR_variable): BOOLEAN;

LOCAL

i: INTEGER;

bounded: SET OF BOOLEAN;

END_LOCAL;


IF (SIZEOF(bounds)>=SIZEOF(variables)) THEN RETURN(FALSE);
END_IF;

REPEAT i:=LOINDEX(variables) TO HIINDEX(variables);

      bounded[i]:=FALSE;

END_REPEAT;

REPEAT i:=LOINDEX(bounds) TO HIINDEX(bounds);

      IF NOT ('NRF_SCHEMA.LITERAL_NUMBER' IN
TYPEOF(bounds[i]\binary_gen_expression.operands[1])) THEN
            RETURN(FALSE);
      END_IF;


      IF NOT ('SIR_PARAMETERIZED_FUNCTION_SCHEMA.SIR_VARIABLE' IN
TYPEOF(bounds[i]\binary_gen_expression.operands[2])) THEN
            RETURN(FALSE);
      END_IF;

END_REPEAT;

RETURN(TRUE);

END_FUNCTION;

FUNCTION ascending_list(values: list_real_literals): BOOLEAN;

LOCAL

      i: INTEGER;
```

```
END_LOCAL;


REPEAT i:= LOINDEX(values) TO (HIINDEX(values)-1);

      IF (values[i]>values[i+1]) THEN
            RETURN(FALSE);
      END_IF;

END_REPEAT;

RETURN(TRUE);

END_FUNCTION;

FUNCTION get_containers(prop: SIR_descriptive_or_functional): SET OF
SIR_list_of_values_or_functions;
LOCAL
      i: INTEGER;
      sav: BAG OF SIR_applicability_of_values;
      result: SET OF SIR_list_of_values_or_functions;
END_LOCAL;

sav:= USEDIN( prop,
'SIR_PROPERT_SCHEMA.SIR_APPLICABILITY_OF_VALUES.VALID_USAGE.PROPERTY');

REPEAT i:=LOINDEX(sav) TO HIINDEX(sav);
      result[i]:= sav[i].valid_data;
END_REPEAT;

RETURN(result);

END_FUNCTION;

TYPE security_classified_item = SELECT (
            SIR_model,
            SIR_node,
            SIR_node_relationship,
            SIR_aspect,
            SIR_ato_campaign,
            SIR_ato_case,
            SIR_ato_phase
            );
END_TYPE;

TYPE dated_item = SELECT (
            SIR_ato_campaign,
            NRF_run
            );
END_TYPE;

TYPE approved_item = SELECT (
            SIR_ato_campaign
            );
END_TYPE;

TYPE sourced_item = SELECT (
            SIR_ato_campaign
            );
END_TYPE;

TYPE named_item = SELECT (
            NRF_run,
            NRF_run_sequence
            );
END_TYPE;
```

```
ENTITY NRF_security_assignment
SUBTYPE OF (security_classification_assignment);
      items: SET[1:?] OF security_classified_item;
WHERE
      WR1:
SELF\security_classification_assignment.assigned_security_classification.security_
level.name IN
['unclassified','classified','proprietary','confidential','secret','top_secret'];
END_ENTITY;

ENTITY NRF_date_assignment
SUBTYPE OF (date_and_time_assignment);
      items: SET[1:?] OF dated_item;
END_ENTITY;

ENTITY NRF_approval_assignment
SUBTYPE OF (approval_assignment);
      items: SET[1:?] OF approved_item;
END_ENTITY;

ENTITY NRF_contact_assignment
SUBTYPE OF (person_and_organization_assignment);
      items: SET[1:?] OF sourced_item;
WHERE
      WR1: SELF\person_and_organization_assignment.role.name='contact_person/
organization';
END_ENTITY;

ENTITY NRF_name_assignment
SUBTYPE OF (name_assignment);
      items: SET[1:?] OF named_item;
END_ENTITY;

ENTITY NRF_model_product_relationship
SUBTYPE OF (property_definition_representation) ;
WHERE
      WR1:
SELF\property_definition_representation.definition.name='product_description_for_a
to';
      WR2: 'NRF_SCHEMA.PRODUCT_DEFINITION' IN
TYPEOF(SELF\property_definition_representation.definition.definition);
      WR3: ('NRF_SCHEMA.SIR_MODEL' IN
TYPEOF(SELF\property_definition_representation.used_representation)) XOR
((SIZEOF(SELF\property_definition_representation.used_representation.items)=1)
AND
(SIZEOF(['NRF_SCHEMA.SIR_NODE','NRF_SCHEMA.SIR_NODE_RELATIONSHIP','NRF_SCHEMA.SIR_
SUBMODEL_USAGE'] *
TYPEOF(SELF\property_definition_representation.used_representation.items[1]))=1));
END_ENTITY;

ENTITY NRF_run
SUBTYPE OF (executed_action);
DERIVE
      run_name:  SET OF label := get_names( SELF);
      creator_tool_or_facility: SET OF action_resource:= get_tools( SELF);
      timestamp: SET OF date_and_time := get_timestamps(SELF);
INVERSE
      subject_association: NRF_run_phase FOR assigned_action;
      results_association: NRF_run_results FOR assigned_action;
UNIQUE
      UR1: SELF\action.name, subject_association;
WHERE
      WR1: SIZEOF(run_name) = 1;
      WR2: SIZEOF(creator_tool_or_facility) = 1;
      WR3: SIZEOF(timestamp) = 1;
END_ENTITY;

ENTITY NRF_run_phase
```

```
SUBTYPE OF (action_assignment);
      subject: SIR_ato_phase;
      SELF\action_assignment.assigned_action: NRF_run;
END_ENTITY;

ENTITY NRF_run_results
SUBTYPE OF (action_assignment);
      results: SET[1:?] OF SIR_aspect;
      SELF\action_assignment.assigned_action: NRF_run;
END_ENTITY;

ENTITY NRF_run_sequence
SUBTYPE OF (action);
DERIVE
      sequence_name:  SET OF label := get_names( SELF);
INVERSE
      subject_association: NRF_run_sequence_case FOR assigned_action;
      sequence: SET[1:?] OF NRF_run_in_sequence FOR relating_action;
UNIQUE
      UR1: SELF\action.name, subject_association;
WHERE
      WR1: SIZEOF(sequence_name) = 1;
END_ENTITY;

ENTITY NRF_run_sequence_case
SUBTYPE OF (action_assignment);
      subject: SIR_ato_case;
      SELF\action_assignment.assigned_action: NRF_run_sequence;
END_ENTITY;

ENTITY NRF_run_in_sequence
SUBTYPE OF (action_relationship);
      SELF\action_relationship.relating_action: NRF_run_sequence;
      SELF\action_relationship.related_action: NRF_run;
WHERE
      WR1: related_action.subject_association.subject.of_case :=:
relating_action.subject_association.subject;
END_ENTITY;

ENTITY NRF_scan_sampled
SUBTYPE OF (SIR_scan);
WHERE
      WR1: SELF\SIR_scan.aspect.lifetime <> invariant;
      WR2: EXISTS(SELF\SIR_scan.abscissa_value);
END_ENTITY;

ENTITY NRF_scan_derived
SUBTYPE OF (SIR_scan);
INVERSE
      derivation: NRF_derivation_result FOR result;
END_ENTITY;

ENTITY NRF_derivation_procedure
SUBTYPE OF (action_method);
UNIQUE
      UR1: SELF\action_method.name;
END_ENTITY;

ENTITY NRF_derivation_result
SUBTYPE OF (action_assignment);
      result: NRF_scan_derived;
      SELF\action_assignment.assigned_action: NRF_derivation;
END_ENTITY;

ENTITY NRF_derivation_bounds
SUBTYPE OF (action_assignment);
      bounds: SET[1:2] OF SIR_number_or_string;
      SELF\action_assignment.assigned_action: NRF_derivation;
```

```
END_ENTITY;

ENTITY NRF_derivation
SUBTYPE OF (executed_action) ;
      SELF\action.chosen_method: NRF_derivation_procedure;
INVERSE
      bounds_association: NRF_derivation_bounds FOR assigned_action;
      result_association: NRF_derivation_result FOR assigned_action;
END_ENTITY;

RULE ato_campaign_requires_creation_date FOR (SIR_ato_campaign,
NRF_date_assignment);
WHERE
      WR1: SIZEOF(QUERY(obj<* SIR_ato_campaign| SIZEOF(QUERY(nda <*
NRF_date_assignment| (obj IN nda.items) AND (nda.role.name = 'creation_date')))<>1
))= 0;
END_RULE;

RULE ato_campaign_requires_contact FOR (SIR_ato_campaign, NRF_contact_assignment);
WHERE
      WR1: SIZEOF(QUERY(obj<* SIR_ato_campaign| SIZEOF(QUERY(nca <*
NRF_contact_assignment | obj IN nca.items))<>1 ))= 0;
END_RULE;

RULE at_most_one_modification_date FOR (SIR_ato_campaign, NRF_date_assignment);
WHERE
      WR1: SIZEOF(QUERY(obj<* SIR_ato_campaign| SIZEOF(QUERY(nda <*
NRF_date_assignment| (obj IN nda.items) AND (nda.role.name =
'last_modification_date')))>1 ))= 0;
END_RULE;

RULE at_most_one_security_level FOR (SIR_model, SIR_node, SIR_node_relationship,
SIR_aspect, SIR_ato_campaign, SIR_ato_case, SIR_ato_phase,
NRF_security_assignment);
WHERE
      WR1: SIZEOF(QUERY(obj<* SIR_model| SIZEOF(QUERY(nsa <*
NRF_security_assignment | obj IN nsa.items))>1 ))= 0;
      WR2: SIZEOF(QUERY(obj<* SIR_node| SIZEOF(QUERY(nsa <* NRF_security_assignment
| obj IN nsa.items))>1 ))= 0;
      WR3: SIZEOF(QUERY(obj<* SIR_node_relationship| SIZEOF(QUERY(nsa <*
NRF_security_assignment | obj IN nsa.items))>1 ))= 0;
      WR4: SIZEOF(QUERY(obj<* SIR_ato_campaign| SIZEOF(QUERY(nsa <*
NRF_security_assignment | obj IN nsa.items))>1 ))= 0;
      WR5: SIZEOF(QUERY(obj<* SIR_ato_case| SIZEOF(QUERY(nsa <*
NRF_security_assignment | obj IN nsa.items))>1 ))= 0;
      WR6: SIZEOF(QUERY(obj<* SIR_ato_phase| SIZEOF(QUERY(nsa <*
NRF_security_assignment | obj IN nsa.items))>1 ))= 0;
END_RULE;

RULE application_context_requires_ap_definition FOR
  (application_context, application_protocol_definition);
WHERE
  WR1: SIZEOF (QUERY (ac <* application_context |
       NOT (SIZEOF (QUERY (apd <* application_protocol_definition |
       ac :=: apd.application)) = 1 ))) = 0;
END_RULE;

RULE approval_requires_approval_date_time FOR (approval,
         approval_date_time);
WHERE
  WR1: SIZEOF (QUERY ( app <* approval |
       NOT (SIZEOF (QUERY (adt <* approval_date_time |
       app :=: adt.dated_approval)) = 1))) = 0;
END_RULE;

RULE approval_date_time_requires_date_time FOR (approval_date_time);
WHERE
  WR1: SIZEOF (QUERY ( app <* approval_date_time |
```

```
      NOT ('NRF_SCHEMA.DATE_AND_TIME' IN TYPEOF(app.date_time)))) = 0;
END_RULE;

RULE approval_requires_approval_person_organization FOR (approval,
        approval_person_organization);
WHERE
  WR1: SIZEOF (QUERY (app <* approval |
       NOT (SIZEOF (QUERY (apo <* approval_person_organization |
       app :=: apo.authorized_approval)) = 1))) = 0;
END_RULE;

RULE approval_person_organization_requires_person_organization FOR
(approval_person_organization);
WHERE
  WR1: SIZEOF (QUERY ( app <* approval_person_organization |
       NOT ('NRF_SCHEMA.PERSON_AND_ORGANIZATION' IN
TYPEOF(app.person_organization)))) = 0;
END_RULE;

RULE approvals_are_assigned FOR
  (approval, NRF_approval_assignment);
WHERE
  WR1: SIZEOF (QUERY (app <* approval |
       NOT (SIZEOF (QUERY (aa <* NRF_approval_assignment |
       app :=: aa.assigned_approval )) >= 1 ))) = 0;
END_RULE;

RULE component_sequence_requires_aspect FOR (SIR_component_sequence);
WHERE
     WR1: SIZEOF(USEDIN(SELF,  'NRF_SCHEMA.SIR_ASPECT.COMPONENT_SEQUENCE')) >0;
END_RULE;

RULE dependent_instantiable_approval_role FOR (approval_role);
WHERE
  WR1: SIZEOF (QUERY (dtr <* approval_role |
       NOT (SIZEOF (USEDIN (dtr, '')) >= 1))) = 0;
END_RULE;

RULE dependent_instantiable_approval_status FOR (approval_status);
WHERE
WR1: SIZEOF (QUERY (ast <* approval_status |
     NOT (SIZEOF (USEDIN (ast, '')) >= 1))) = 0;
END_RULE;

RULE dependent_instantiable_date FOR (date);
WHERE
     WR1: SIZEOF (QUERY (dt <* date |
      NOT (SIZEOF(USEDIN (dt, 'NRF_SCHEMA.DATE_AND_TIME.DATE_COMPONENT')) >= 1)))
= 0;
END_RULE;

RULE dependent_instantiable_date_time_role FOR (date_time_role);
WHERE
  WR1: SIZEOF (QUERY (dtr <* date_time_role |
       NOT (SIZEOF (USEDIN (dtr, '')) >= 1))) = 0;
END_RULE;

RULE dependent_instantiable_person FOR (person);
WHERE
     WR1: SIZEOF(USEDIN(SELF,  'NRF_SCHEMA.PERSON_AND_ORGANIZATION.THE_PERSON'))
>0;
END_RULE;

RULE dependent_instantiable_person_and_organization_role FOR (
        person_and_organization_role);
WHERE
  WR1: SIZEOF (QUERY (poar <* person_and_organization_role |
       NOT (SIZEOF (USEDIN (poar, '')) >= 1))) = 0;
```

```
END_RULE;

RULE dependent_instantiable_security_classification_level FOR
(security_classification_level);
WHERE
  WR1: SIZEOF (QUERY (scl <* security_classification_level |
       NOT (SIZEOF (USEDIN (scl, '')) >= 1))) = 0;
END_RULE;

RULE dependent_instantiable_type_qualifier FOR (
type_qualifier);
WHERE
  WR1: SIZEOF (QUERY (poar <* type_qualifier|
       NOT (SIZEOF (USEDIN (poar, '')) >= 1))) = 0;
END_RULE;

RULE exclusive_subtypes_action_assignment FOR (action_assignment);
WHERE
      WR1: SIZEOF (QUERY (act <* action_assignment |
SIZEOF(['NRF_SCHEMA.NRF_RUN_PHASE','NRF_SCHEMA.NRF_RUN_RESULTS','NRF_SCHEMA.NRF_RU
N_SEQUENCE_CASE', 'NRF_SCHEMA.NRF_DERIVATION_RESULT',
'NRF_SCHEMA.NRF_DERIVATION_BOUNDS']*TYPEOF(act))<>1)) = 0;
END_RULE;

RULE exclusive_subtypes_of_scan FOR (SIR_scan) ;
WHERE
      WR1: SIZEOF(QUERY(ss <* SIR_scan |
 ((('NRF_SCHEMA.NRF_SCAN_DERIVED') IN TYPEOF(ss)) AND
(('NRF_SCHEMA.NRF_SCAN_SAMPLED') IN TYPEOF(ss))) ))  =0;
END_RULE;

RULE product_requires_version FOR (product, SIR_product_version);
WHERE
  WR1: SIZEOF (QUERY (prod <* product |
       NOT (SIZEOF (QUERY (pdf <* SIR_product_version |
       prod :=: pdf\product_definition_formation.of_product )) >= 1 ))) = 0;
END_RULE;

RULE model_requires_definition FOR (SIR_model, NRF_model_product_relationship);
WHERE
      WR1: SIZEOF(QUERY(sm <* SIR_model | SIZEOF(QUERY(nmpr <*
NRF_model_product_relationship | sm :=:
nmpr\property_definition_representation.used_representation))=0))=0;
END_RULE;

RULE subtype_mandatory_action FOR (action);
WHERE
      WR1: SIZEOF (QUERY (act <* action |
SIZEOF(['NRF_SCHEMA.NRF_RUN','NRF_SCHEMA.NRF_RUN_SEQUENCE',
'NRF_SCHEMA.NRF_DERIVATION']*TYPEOF(act))<>1)) = 0;
END_RULE;

RULE subtype_mandatory_action_relationship FOR (action_relationship);
WHERE
      WR1: SIZEOF (QUERY (act <* action_relationship |
NOT ('NRF_SCHEMA.NRF_RUN_IN_SEQUENCE' IN TYPEOF(act)))) = 0;
END_RULE;

RULE subtype_mandatory_date FOR (date);
WHERE
      WR1: SIZEOF (QUERY (act <* date|NOT('NRF_SCHEMA.CALENDAR_DATE' IN
TYPEOF(act)))) = 0;
END_RULE;

RULE subtype_mandatory_product_definition_formation FOR
(product_definition_formation);
WHERE
```

```
      WR1: SIZEOF(QUERY(obj<*
product_definition_formation|NOT('NRF_SCHEMA.SIR_PRODUCT_VERSION' IN
TYPEOF(obj))))= 0;
END_RULE;

RULE subtype_mandatory_product_definition_relationship FOR
(product_definition_relationship);
WHERE
      WR1: SIZEOF(QUERY(obj<*
product_definition_relationship|NOT('NRF_SCHEMA.NEXT_ASSEMBLY_USAGE_OCCURENCE' IN
TYPEOF(obj))))= 0;
END_RULE;

RULE unique_ids_in_model FOR (SIR_model, SIR_node, SIR_node_relationship,
SIR_submodel_usage);
WHERE
      WR1:SIZEOF(QUERY(sm <* SIR_model | NOT (unique_ids(QUERY(sn<* SIR_node | sn
IN sm\representation.items)) AND unique_ids(QUERY(snr<* SIR_node_relationship |
snr IN sm\representation.items)) AND unique_ids(QUERY(su<* SIR_submodel_usage | su
IN sm\representation.items))) )) =0;
END_RULE;

FUNCTION get_names(prop: named_item): SET OF label;
LOCAL
      i: INTEGER;
      sav: BAG OF NRF_name_assignment;
      result: SET OF label;
END_LOCAL;

sav := USEDIN( prop, 'NRF_SCHEMA.NRF_NAME_ASSIGNMENT.ITEMS');

REPEAT i:=LOINDEX(sav) TO HIINDEX(sav);
      result[i]:=sav[i].assigned_name;
END_REPEAT;

RETURN(result);
END_FUNCTION;

FUNCTION get_tools(prop: NRF_run): SET OF action_resource;
LOCAL
      i: INTEGER;
      sav: BAG OF action_resource;
      result: SET OF action_resource;
END_LOCAL;

sav := USEDIN( prop, 'NRF_SCHEMA.ACTION_RESOURCE.USAGE');

REPEAT i:=LOINDEX(sav) TO HIINDEX(sav);
      IF (sav[i].kind.name='tool_or_facility') THEN
            result[i]:=sav[i];
      END_IF;
END_REPEAT;

RETURN(result);
END_FUNCTION;

FUNCTION get_timestamps(prop: NRF_run): SET OF date_and_time;
LOCAL
      i: INTEGER;
      sav: BAG OF NRF_date_assignment;
      result: SET OF date_and_time;
END_LOCAL;

sav := USEDIN( prop, 'NRF_SCHEMA.NRF_DATE_ASSIGNMENT.ITEMS');

REPEAT i:=LOINDEX(sav) TO HIINDEX(sav);
      IF (sav[i].role.name='start_date_and_time') THEN
            result[i]:=sav[i].assigned_date_and_time;
```

```
      END_IF;
END_REPEAT;

RETURN(result);
END_FUNCTION;

FUNCTION unique_ids(sav: BAG OF SIR_model_constituent): LOGICAL;
LOCAL
      i: INTEGER;
      bag_id: BAG OF identifier;
END_LOCAL;

REPEAT i:=LOINDEX(sav) TO HIINDEX(sav);
      bag_id[i] := sav[i].id;
END_REPEAT;

RETURN(VALUE_UNIQUE(bag_id));
END_FUNCTION;



END_SCHEMA;
(*
```

Annex B - AIM short names of entities (normative)

This annex provides a unique assigned acronym for each entity name occuring in this protocol. The long form entity names have been extracted from the EXPRESS language statements found in the AIM EXPRESS listing of annex A.

Requirements on the use of the short names are found in the implementation methods included in ISO 10303.

Long Name	*Short Name*
action	ACTION
action_assignment	ACTASS
action_method	ACTMTH
action_relationship	ACTRLT
action_resource	ACTRSR
action_resource_type	ACRSTY
address	ADDRSS
amount_of_substance_measure_with_unit	AOSMWU
amount_of_substance_unit	AOSU
application_context	APPCNT
application_context_element	APCNEL
approval	APPRVL
approval_assignment	APPASS
approval_date_time	APDTTM
approval_person_organization	APPROR
approval_role	APPRL
approval_status	APPSTT
application_protocol_definition	APPRDF
area_measure_with_unit	AMWU
area_unit	ARUNT
assembly_component_usage	ASCMUS
calendar_date	CLNDT
context_dependent_unit	CNDPUN
conversion_based_unit	CNBSUN
coordinated_universal_time_offset	CUTO
date	DATE
date_and_time	DTANTM
date_and_time_assignment	DATA
date_time_role	DTTMRL
derived_unit	DRVUNT
derived_unit_element	DRUNEL
dimensional_exponents	DMNEXP
document	DCMNT
document_type	DCMTYP
document_usage_constraint	CNBSUN
electric_current_measure_with_unit	ECMWU

Long Name	*Short Name*
electric_current_unit	ELCRUN
executed_action	EXCACT
group	GROUP
group_assignment	GRPASS
length_measure_with_unit	LMWU
length_unit	LNGUNT
local_time	LCLTM
luminous_intensity_measure_with_unit	LIMWU
luminous_intensity_unit	LMINUN
mapped_item	MPPITM
mass_measure_with_unit	MMWU
mass_unit	MSSUNT
name_assignment	NMASS
named_unit	NMDUNT
next_assembly_usage_occurence	NAUO
NRF_approval_assignment	NRFAA
NRF_contact_assignment	NRFCA
NRF_date_assignment	NRFDA
NRF_derivation	NRFD
NRF_derivation_bounds	NRFDB
NRF_derivation_procedure	NRFDP
NRF_derivation_result	NRFDR
NRF_model_product_relationship	NRFMPR
NRF_name_assignment	NRFNA
NRF_run	NRFR
NRF_run_in_sequence	NRFRIS
NRF_run_phase	NRFRP
NRF_run_results	NRFRR
NRF_scan_derived	NRFSD
NRF_scan_sampled	NRFSS
NRF_security_assignment	NRFSA
NRF_sequence	NRFS
NRF_sequence_case	NRFSC
organization	ORGNZT
organizational_address	ORGADD
organizational_project	ORGPRJ
person	PERSON
person_and_organization	PRANOR
person_and_organization_assignment	PAOA
person_and_organization_role	PAOR
personal_address	PRSADD
plane_angle_measure_with_unit	PAMWU
plane_angle_unit	PLANUN
product	PRDCT
product_context	PRDCNT
product_definition	PRDDFN
product_definition_context	PRDFCN

Long Name	*Short Name*
product_definition_formation	PRDFFR
product_definition_relationship	PRDFRL
product_definition_usage	PRDFUS
property_definition	PRPDFN
property_definition_representation	PRDFRP
ratio_measure_with_unit	RMWU
ratio_unit	RTUNT
representation	RPRSNT
representation_context	RPRCNT
representation_item	RPRITM
representation_map	RPRMP
security_classification	SCRCLS
security_classification_assignment	SCCLAS
security_classification_level	SCCLLV
si_unit	SUNT
SIR_applicability_of_values	SIRAOV
SIR_aspect	SIRA
SIR_ato_campaign	SIRCAM
SIR_ato_case	SIRCAS
SIR_ato_phase	SIRP
SIR_case_in_campaign	SIRCIC
SIR_component_sequence	SIRCS
SIR_cyclic_tabular_function	SIRCTF
SIR_index_interval	SIRII
SIR_initial_conditioning	SIRIC
SIR_list_of_descriptive_values	SIRLDV
SIR_list_of_functions	SIRLOF
SIR_mask	SIRMA
SIR_model	SIRMO
SIR_node	SIRN
SIR_node_relationship	SIRNR
SIR_node_relationship_usage	SIRNRU
SIR_node_usage	SIRNU
SIR_parameterized_function	SIRPAF
SIR_polynomial_function	SIRPOF
SIR_product_version	SIRPV
SIR_property_class	SIRPC
SIR_property_descriptive	SIRPD
SIR_property_functional	SIRPF
SIR_property_meaning	SIRPM
SIR_property_name	SIRPN
SIR_property_quantitative	SIRPQ
SIR_property_scalar	SIRPS
SIR_property_tensor	SIRPT
SIR_property_usage	SIRPU
SIR_scalar_in_tensor	SIRSIT
SIR_scan	SIRS

Long Name	*Short Name*
SIR_submodel_usage	SIRSU
SIR_tabular_function	SIRTF
SIR_unit_assignment	SIRUA
SIR_variable	SIRV
solid_angle_measure_with_unit	SAMWU
solid_angle_unit	SLANUN
thermodynamic_temperature_measure_unit	TTMWU
thermodynamic_temperature_unit	THTMUN
time_measure_with_unit	TMWU
time_unit	TMUNT
type_qualifier	TYPQLF
volume_unit	VLMUNT
binary_gen_expression	ISO13584_BIGEEX
binary_n_expression	ISO13584_BINEX
boolean_expression	ISO13584_BOOEXP
comparison_expression	ISO13584_COMEXP
comparison_greater_equal	ISO13584_COGREQ
comparison_less_equal	ISO13584_CLOE
environment	ISO13584_ENVIRO
expression	ISO13584_EXPRES
generic_expression	ISO13584_GENEXP
generic_literal	ISO13584_GENLIT
generic_variable	ISO13584_GENVAR
int_literal	ISO13584_INTLIT
int_numeric_variable	ISO13584_INNUVA
literal_number	ISO13584_LITNUM
mathematical_string	ISO13584_MATSTR
mult_expression	ISO13584_MULEXP
m_ary_gen_expression	ISO13584_MAGE
m_ary_n_expression	ISO13584_MANE
numeric_expression	ISO13584_NUMEXP
numeric_variable	ISO13584_NUMVAR
plus_expression	ISO13584_PLUEXP
power_expression	ISO13584_POWEXP
real_literal	ISO13584_REALIT
real_numeric_variable	ISO13584_RENUVA
simple_gen_expresssion	ISO13584_SIGEEX
unary_gen_expression	ISO13584_UNGEEX
unary_n_expression	ISO13584_UNNEX
simple_n_expression	ISO13584_SINEX
variable	ISO13584_VARIAB
variable_semantics	ISO13584_VARSEM

Annex C - Implementation method specific requirements (normative)

An implementation method defines the exchange characteristics and representation that are required by this part. Conformance to this part shall be realized using one of the implementation methods specified in ISO 10303.

This annex specifies additional requirements for the use of this part of ISO 10303 with an exchange structure as specified in ISO 10303-21.

C.1 Reference to the AIM schema

This annex specifies requirements on the header section of an exchange structure.

The attribute **schema_identifiers** of the entity **file_name** of the header section of an exchange structure shall for the purpose of this part include in its list of **schema_names** the name '**NRF_SCHEMA**' and only this name.

Annex G - Application Reference Model (Informative)

G.1 EXPRESS definition of the ARM application objects

```
SCHEMA nrf_arm_schema;

-- Annex to NRF-002-AP release 1.4, 15 November 1996

-- Checked with ST-EXPRESS by STEPTools Inc.
-- and IS versions of EXPRESS schemas for ISO 10303 parts 41, 42, 43

USE FROM support_resource_schema (
  identifier,
  label,
  text );                                        -- defined in ISO 10303 part 41

USE FROM date_time_schema;
                                                 -- defined in ISO 10303 part 41

USE FROM measure_schema;
                                                 -- defined in ISO 10303 part 41

USE FROM person_organization_schema;
                                                 -- defined in ISO 10303 part 41

USE FROM document_schema;
                                                 -- defined in ISO 10303 part 41

USE FROM security_classification_schema (
  security_classification_level );               -- defined in ISO 10303 part 41

TYPE abscissa_sequencing_type = ENUMERATION OF (
  strictly_decreasing,
  monotonic_decreasing,
  monotonic_increasing,
  strictly_increasing);
END_TYPE;

(* address used from ISO-10303-part-41 person_organization_schema
 * ENTITY address;
 *   internal_location       : OPTIONAL label;
 *   street_number           : OPTIONAL label;
 *   street                  : OPTIONAL label;
 *   postal_box              : OPTIONAL label;
 *   town                    : OPTIONAL label;
 *   region                  : OPTIONAL label;
 *   postal_code             : OPTIONAL label;
 *   country                 : OPTIONAL label;
 *   facsimile_number        : OPTIONAL label;
 *   telephone_number        : OPTIONAL label;
 *   electronic_mail_address : OPTIONAL label;
 *   telex_number            : OPTIONAL label;
 * WHERE
 *   WR1: EXISTS(internal_location)       OR
 *        EXISTS(street_number)           OR
 *        EXISTS(street)                  OR
 *        EXISTS(postal_box)              OR
 *        EXISTS(town)                    OR
 *        EXISTS(region)                  OR
 *        EXISTS(postal_code)             OR
 *        EXISTS(country)                 OR
 *        EXISTS(facsimile_number)        OR
 *        EXISTS(telephone_number)        OR
 *        EXISTS(electronic_mail_address) OR
 *        EXISTS(telex_number);
```

```
 * END_ENTITY;
 *)

ENTITY approval;
  -- is a simplification of approval in ISO-10303-41 approval_schema
  level                 : label;
  date_time             : date_and_time;
  by_person_organization : person_and_organization;
END_ENTITY;

ENTITY ato_aspect;
  name                  : label;
  for_property_class    : property_class_usage;
  lifetime              : lifetime_type;
  component_sequence    : model_component_sequence;
  scans                 : LIST [1:?] OF scan;
  security_class        : OPTIONAL security_classification_level;
END_ENTITY;

ENTITY ato_campaign;
  -- "ato" stands for "analysis, test or operation"
  of_project                      : organizational_project;
  name                            : label;
  description                     : text;
  creation_date_and_time          : date_and_time ;
  last_modification_date_and_time : OPTIONAL date_and_time ;
  contact_person_organization     : person_and_organization;
  approvals                       : SET [0:?] OF approval;
  security_class                  : OPTIONAL security_classification_level;
  arm_schema_name                 : STRING;
  arm_protocol_year               : INTEGER;
UNIQUE
  ur1 : of_project, name;
END_ENTITY;

ENTITY ato_case;
  -- "ato" stands for "analysis, test or operation"
  of_campaign         : ato_campaign;
  id                  : identifier;
  name                : label;
  description         : text;
  root_model          : network_model;
  abscissa_class      : property_class_usage;
  abscissa_sequencing : OPTIONAL abscissa_sequencing_type;
  predefined_aspects  : SET [0:?] OF ato_aspect;
  security_class      : OPTIONAL security_classification_level;
INVERSE
  phases              : SET [0:?] OF ato_phase FOR of_case;
  unit_assignments    : SET [1:?] OF unit_assignment FOR of_case;
UNIQUE
  ur1 : of_campaign, id;
END_ENTITY;

ENTITY ato_phase;
  -- "ato" stands for "analysis, test or operation"
  of_case            : ato_case;
  id                 : identifier;
  name               : label;
  description        : text;
  predefined_aspects : SET [0:?] OF ato_aspect;
  security_class     : OPTIONAL security_classification_level;
UNIQUE
  ur1 : of_case, id;
END_ENTITY;

(* calendar_date used from ISO-10303-41 date_time_schema
 * ENTITY calendar_date
 *  SUBTYPE OF (date);
```

```
 *   day_component   : day_in_month_number;
 *   month_component : month_in_year_number;
 * WHERE
 *   WR1: valid_calendar_date (SELF);
 * END_ENTITY;
 *)

ENTITY cyclic_tabular_function
  SUBTYPE OF ( tabular_function );
  period : scalar_value;
WHERE
  wr1 : SIZEOF( SELF\parameterized_function.parameter_properties ) = 1;
END_ENTITY;

(* date used from ISO-10303-41 date_time_schema
 * ENTITY date
 *   SUPERTYPE OF (ONEOF (calendar_date,
 *                        ordinal_date,
 *                        week_of_year_and_day_date));
 *   year_component : year_number;
 * END_ENTITY;
 *)

(* date_and_time used from ISO-10303-41 date_time_schema
 * ENTITY date_and_time;
 *  date_component : date;
 *  time_component : local_time;
 * END_ENTITY;
 *)

(* document used from ISO-10303-41 document_schema
 * ENTITY document;
 *   id          : identifier;
 *   name        : label;
 *   description : text;
 *   kind        : document_type;
 * UNIQUE
 *   UR1: id;
 * END_ENTITY;
 *)

(* document_type used from ISO-10303-41 document_schema
 * ENTITY document_type;
 *   name : label; -- Modified! Attribute was called product_data_type in ISO-
10303-41
 * END_ENTITY;
 *)

(* document_usage_constraint type used from ISO-10303-41 document_schema
 * ENTITY document_usage_constraint;
 *   source               : document;
 *   subject_element      : label;
 *   subject_element_value : text;
 * END_ENTITY;
 *)

TYPE lifetime_type = ENUMERATION OF (
  invariant,
  sample,
  interval ) ;
END_TYPE;

ENTITY list_of_descriptive_values;
  applicable_for : SET [1:?] OF property_class_usage;
  name           : label;
  entries        : LIST [1:?] OF STRING;
END_ENTITY;
```

```
ENTITY list_of_functions;
  applicable_for : SET [1:?] OF property_class_usage;
  name           : label;
  entries        : LIST [1:?] OF parameterized_function;
END_ENTITY;

(* local_time used from ISO-10303-part-41 date_time_schema
 * ENTITY local_time;
 *   hour_component   : hour_in_day;
 *   minute_component : OPTIONAL minute_in_hour;
 *   second_component : OPTIONAL second_in_minute;
 *   zone             : coordinated_universal_time_offset;
 * WHERE
 *   WR1: valid_time (SELF);
 * END_ENTITY;
 *)

ENTITY model_component_sequence;
  id         : identifier;
  name       : label;
  components : LIST [1:?] OF UNIQUE network_model_component;
DERIVE
  root_model : network_model := find_root_model( components );
END_ENTITY;

FUNCTION find_root_model( mcl : LIST OF network_model_component ) : network_model;
-- to be implemented
END_FUNCTION;

ENTITY model_represents_product_relationship;
  model_constituent   : network_model_or_constituent;
  represented_product : product_definition;
END_ENTITY;

ENTITY network_model;
  id             : identifier;
  version_id     : identifier;
  name           : label;
  description    : text;
  security_class : OPTIONAL security_classification_level;
  constituents   : SET [1:?] OF network_model_constituent;
UNIQUE
  ur1: id, version_id;
END_ENTITY;

TYPE network_model_component = SELECT (
  network_model,
  network_node,
  network_node_relationship,
  submodel_usage,
  network_node_usage,
  network_node_relationship_usage );
END_TYPE;

TYPE network_model_constituent = SELECT (
  network_node,
  network_node_relationship,
  submodel_usage );
END_TYPE;

TYPE network_model_or_constituent = SELECT (
  network_model,
  network_model_constituent );
END_TYPE;

ENTITY network_node;
  id             : identifier;
  name           : OPTIONAL label;
```

```
  class          : LIST [1:?] OF network_node_class;
  security_class : OPTIONAL security_classification_level;
END_ENTITY;

ENTITY network_node_class;
  name  : label;
UNIQUE
  ur1: name;
END_ENTITY;

TYPE network_node_select = SELECT (
  network_node,
  network_node_usage );
END_TYPE;

ENTITY network_node_relationship;
  id             : identifier;
  name           : OPTIONAL label;
  class          : LIST [1:?] OF network_node_relationship_class;
  nodes          : LIST [2:?] OF network_node_select;
  security_class : OPTIONAL security_classification_level;
END_ENTITY;

ENTITY network_node_relationship_class;
  name  : label;
UNIQUE
  ur1: name;
END_ENTITY;

ENTITY network_node_relationship_usage;
  used_submodel : submodel_usage;
  relationship  : network_node_relationship;
END_ENTITY;

ENTITY network_node_usage;
  used_submodel : submodel_usage;
  node          : network_node;
END_ENTITY;

(* organization used from ISO-10303-41 person_organization_schema
 * ENTITY organization;
 *   id          : OPTIONAL identifier;
 *   name        : label;
 *   description : text;
 * END_ENTITY;
 *)

(* organizational_address used from ISO-10303-41 person_organization_schema
 * ENTITY organizational_address
 *   SUBTYPE OF (address);
 *   organizations : SET [1:?] OF organization;
 *   description   : text;
 * END_ENTITY;
 *)

(* organizational_project used from ISO-10303-41 person_organization_schema
 * ENTITY organizational_project;
 *   name                      : label;
 *   description               : text;
 *   responsible_organizations : SET[1:?] OF organization;
 * END_ENTITY;
 *)

ENTITY parameterized_function
  ABSTRACT SUPERTYPE OF( ONEOF(
    parameterized_function_with_formula,
    polynomial_function,
    tabular_function ) );
```

```
  name                : label;
  description         : text;
  parameter_properties : LIST [0:?] OF property_class_scalar_quantitative;
  result_property     : property_class_scalar_quantitative;
  source              : OPTIONAL document_usage_constraint;
  definition_timestamp : OPTIONAL date_and_time;
END_ENTITY;

ENTITY parameterized_function_with_formula
  SUBTYPE OF( parameterized_function );
  formula                 : text;
  language                : OPTIONAL label;
  creator_tool_or_facility : OPTIONAL label;
END_ENTITY;

(* person used from ISO-10303-41 person_organization_schema
 * ENTITY person;
 *   id           : identifier;
 *   last_name    : OPTIONAL label;
 *   first_name   : OPTIONAL label;
 *   middle_names  : OPTIONAL LIST [1:?] OF label;
 *   prefix_titles : OPTIONAL LIST [1:?] OF label;
 *   suffix_titles : OPTIONAL LIST [1:?] OF label;
 * UNIQUE
 *   UR1: id;
 * WHERE
 *   WR1: EXISTS(last_name) OR EXISTS(first_name);
 * END_ENTITY;
 *)

(* person_and_organization used from ISO-10303-41 person_organization_schema
 * ENTITY person_and_organization;
 *   the_person       : person;
 *   the_organization : organization;
 * END_ENTITY;
 *)

(* personal_address used from ISO-10303-41 person_organization_schema
 * ENTITY personal_address
 *   SUBTYPE OF (address);
 *   people      : SET [1:?] OF person;
 *   description : text;
 * END_ENTITY;
 *)

ENTITY polynomial_function
  SUBTYPE OF( parameterized_function );
  degree               : INTEGER;
  coefficients         : LIST [1:?] OF REAL;
  lower_domain_bounds  : ARRAY [1:3] OF OPTIONAL REAL;
  upper_domain_bounds  : ARRAY [1:3] OF OPTIONAL REAL;
WHERE
  wr1: (degree >= 1) AND (degree <= 10);
END_ENTITY;

ENTITY product;
  id                 : identifier;
  name               : label;
  description        : text;
  frame_of_reference : SET [1:?] OF product_context;
END_ENTITY;

ENTITY product_context;
  name            : label;
  discipline_type : label;
END_ENTITY;

ENTITY product_definition;
```

```
  id                : identifier;
  description       : text;
  frame_of_reference : product_definition_context;
UNIQUE
  ur1: id, frame_of_reference;
END_ENTITY;

ENTITY product_definition_context;
  name             : label;
  life_cycle_stage : label;
END_ENTITY;

ENTITY product_next_assembly_usage_relationship;
  id          : identifier;
  assembly    : product_definition;
  constituent : product_definition;
UNIQUE
  ur1: id, assembly;
END_ENTITY;

ENTITY product_version;
  of_product  : product;
  id          : identifier;
  description : text;
  definitions : SET [1:?] OF product_definition;
UNIQUE
  ur1: id, of_product;
END_ENTITY;

TYPE property_class = SELECT (
  property_class_scalar,
  property_class_tensor );
END_TYPE;

ENTITY property_class_scalar
  ABSTRACT SUPERTYPE OF ( ONEOF (
    property_class_scalar_descriptive,
    property_class_scalar_functional,
    property_class_scalar_quantitative ) );
  name           : property_name;
  symmetry       : OPTIONAL property_symmetry;
END_ENTITY;

ENTITY property_class_scalar_descriptive
  SUBTYPE OF ( property_class_scalar );
DERIVE
  value_container : SET [1:?] OF list_of_descriptive_values
    := find_list_of_descriptive_values( SELF );
END_ENTITY;

FUNCTION find_list_of_descriptive_values(
  pcsd : property_class_scalar_descriptive ) : SET OF list_of_descriptive_values;
-- to be implemented
END_FUNCTION;

ENTITY property_class_scalar_functional
  SUBTYPE OF ( property_class_scalar );
DERIVE
  function_container : list_of_functions
    := find_list_of_functions( SELF );
END_ENTITY;

FUNCTION find_list_of_functions(
  pcsf : property_class_scalar_functional ) : SET OF list_of_functions;
-- to be implemented
END_FUNCTION;

ENTITY property_class_scalar_quantitative
```

```
  SUBTYPE OF ( property_class_scalar );
END_ENTITY;

ENTITY property_class_tensor;
  name           : property_name;
  tensor_order   : INTEGER;
  dimensionality : INTEGER;
INVERSE
  members        : SET [1:?] OF scalar_in_tensor FOR tensor;
WHERE
  wr1: (   (tensor_order = 1) AND (dimensionality >= 2)      ) OR
       ( ( (tensor_order = 2) AND (dimensionality =  2) ) OR
         ( (tensor_order = 2) AND (dimensionality =  3) )    ) OR
       ( ( (tensor_order = 4) AND (dimensionality =  2) ) OR
         ( (tensor_order = 4) AND (dimensionality =  3) )    ) ;
END_ENTITY;

ENTITY property_class_usage;
  of_case : ato_case;
  property : property_class;
  meaning  : LIST [1:?] OF property_role;
END_ENTITY;

ENTITY property_name;
  name : label;
UNIQUE
  ur1 : name;
END_ENTITY;

ENTITY property_role;
  name : label;
UNIQUE
  ur1 : name;
END_ENTITY;

TYPE property_symmetry = ENUMERATION OF (
  symmetrical,
  antisymmetrical );
END_TYPE;

ENTITY run;
  of_phase                : ato_phase;
  id                      : identifier;
  name                    : label;
  description             : text;
  timestamp               : date_and_time;
  creator_tool_or_facility : OPTIONAL label;
  aspects                 : SET [1:?] OF ato_aspect;
UNIQUE
  ur1: of_phase, id;
END_ENTITY;

ENTITY run_sequence;
  of_case     : ato_case;
  id          : identifier;
  name        : label;
  description : text;
  runs        : LIST [1:?] OF run;
UNIQUE
  ur1 : of_case, id;
END_ENTITY;

ENTITY scalar_in_tensor;
  tensor   : property_class_tensor;
  scalar   : property_class_scalar;
  position : INTEGER;
  roles    : LIST [1:?] OF property_role;
END_ENTITY;
```

```
TYPE scalar_value = NUMBER ;
END_TYPE;

ENTITY scan
  ABSTRACT SUPERTYPE OF( ONEOF(
    scan_of_sampled_values,
    scan_of_derived_values ) );
  abscissa_value   : OPTIONAL scalar_value;
  mask             : OPTIONAL scan_mask;
  number_of_values : INTEGER;
  values           : ARRAY [1:number_of_values] OF OPTIONAL scalar_value;
INVERSE
  of_aspect        : ato_aspect FOR scans;
END_ENTITY;

ENTITY scan_derivation_procedure;
  name : label;
UNIQUE
  ur1 : name;
END_ENTITY;

ENTITY scan_mask;
  defined_ranges : LIST [1:?] OF ARRAY [1:2] OF INTEGER;
END_ENTITY;

ENTITY scan_of_derived_values
  SUBTYPE OF ( scan );
  derivation_procedure : scan_derivation_procedure;
  abscissa_start       : OPTIONAL scalar_value;
  abscissa_end         : OPTIONAL scalar_value;
END_ENTITY;

ENTITY scan_of_sampled_values
  SUBTYPE OF ( scan );
WHERE
  wr1: EXISTS( SELF\scan.abscissa_value );
END_ENTITY;

(* security_classification_level used from
 * ISO 10303-21 security_classification_schema
 *
 * ENTITY security_classification_level;
 *   name : label;
 * END_ENTITY;
 *)

(*
 * restrict_security_classification_level copied from ISO-10303-203
 *)
RULE restrict_security_classification_level FOR (security_classification_level);
WHERE
  WR1: SIZEOF (QUERY (scl <* security_classification_level |
       NOT (scl.name IN ['unclassified', 'classified', 'proprietary',
       'confidential', 'secret', 'top_secret']))) = 0;
END_RULE; -- restrict_security_classification_level

ENTITY submodel_usage;
  id          : identifier;
  name        : label;
  submodel    : network_model;
  source_node : network_node;
  target_node : network_node;
END_ENTITY;

ENTITY tabular_function
  SUBTYPE OF ( parameterized_function );
  parameter_values     : LIST [1:?] OF scalar_value;
```

```
  function_values      : LIST [1:?] OF scalar_value;
  interpolation_type   : tabular_interpolation_type;
  interpolation_degree : OPTIONAL INTEGER;
END_ENTITY;

TYPE tabular_interpolation_type = ENUMERATION OF (
  linear_logarithmic,
  polynomial );
END_TYPE;

(* unit used from ISO-10303-41 measure_schema
 * Here some additional ENTITY definitions are given to provide
 * insight on how units are defined in ISO-10303-41.
 *
 * TYPE unit = SELECT
 *   (named_unit,
 *    derived_unit);
 * END_TYPE;
 *
 * ENTITY named_unit
 *   SUPERTYPE OF (ONEOF (si_unit, conversion_based_unit, context_dependent_unit)
 *                 ANDOR
 *                 ONEOF (length_unit,
 *                        mass_unit,
 *                        time_unit,
 *                        electric_current_unit,
 *                        thermodynamic_temperature_unit,
 *                        amount_of_substance_unit,
 *                        luminous_intensity_unit,
 *                        plane_angle_unit,
 *                        solid_angle_unit,
 *                        area_unit,
 *                        volume_unit,
 *                        ratio_unit ));
 *   dimensions : dimensional_exponents;
 * END_ENTITY;
 *
 * ENTITY derived_unit;
 *   elements : SET [1:?] OF derived_unit_element;
 * WHERE
 *   WR1 : ( SIZEOF ( elements ) > 1 ) OR
 *           (( SIZEOF ( elements ) = 1 ) AND ( elements[1].exponent <> 1.0 ));
 * END_ENTITY;
 *
 * ENTITY derived_unit_element;
 *   unit     : named_unit;
 *   exponent : REAL;
 * END_ENTITY;
 *)

ENTITY unit_assignment;
  of_case            : ato_case;
  for_property_class : property_class_scalar;
  assigned_unit      : unit;
UNIQUE
  ur1: of_case, for_property_class;
END_ENTITY;

END_SCHEMA; -- nrf_arm_schema
```

G.2 EXPRESS-G Diagram of the ARM

The EXPRESS-G diagrams of the ARM are supplied as an annex of 18 single sided A4-pages, that can be assembled into one large diagram of 6x3 pages.

Annex H - AIM EXPRESS-G (informative)

The following diagrams correspond to the short form of the AIM schema, defined in clause 5.2. The figures use the EXPRESS-G graphical notation for the EXPRESS language.

EXPRESS-G is defined in annex D of ISO 10303-11.

The existence of global rules on an entity are indicated with the asterisk notation of the language.

Uniqueness and local rules are not indicated.

The EXPRESS-G diagrams of the AIM are supplied as an annex of 44 single sided A4-pages, that can be assembled into one large diagram of 11x4 pages.

Annex J - AIM EXPRESS Listing (Informative)

Supplied as file l_form.DOS on a MS-DOS formatted 1.44 MB DS/HD floppy disk.

Annex K - Application protocol implementation and usage guide (informative)

K.1 Coefficients of a polynomial function

Polynomial functions of a given degree and a given number of parameters usually do not contain as much as monoms as there are element in the relevant base of polynoms.

The proposed mapping enables to transfer only the non-zero monoms.

In order to limit the number of instances needed to transfer a polynomial function, it is advised, therefore, not to transfer zero coefficients.

Annex M - Bibliography (Informative)

M.1 ISO 10303 related documents

[STEP-AP-Guide] Guidelines for the development and approval of STEP application protocols, version 1.2, M. Palmer, M. Gilbert, ISO TC184/SC4 N512, 5 April 1996.

[STEP-map-Guide] Guidelines for the development of mapping tables, A. Barnard, D. Craig, ISO TC184/SC4 N436, 5 April 1996.

[STEP-AIM-Guide] Guidelines for AIM development, A. Barnard, M. Gilbert, ISO TC184/SC4 N435, 5 April 1996.

[STEP-Sup-Dir] Supplementary directives for the drafting and presentation of ISO 10303, Joan Wellington, Nigel Shaw ISO TC184/SC4 N432, 5 April 1996.

[STEP-1] ISO 10303-1 (IS)
Part 1: Overview and Fundamental Principles

[STEP-11] ISO 10303-11 (IS)
Part 11: The EXPRESS Language Reference Manual

[STEP-21] ISO 10303-21 (IS)
Part 21: Clear Text Encoding of the Exchange Structure

[STEP-22] ISO 10303-22 (CD)
Part 22: Standard Data Access Interface (SDAI)

[STEP-41] ISO 10303-41 (IS)
Part 41: Integrated Generic Resources: Fundamentals of Product Description and Support

[STEP-43] ISO 10303-43 (IS)
Part 43: Integrated Resources: Representation Structures

[STEP-44] ISO 10303-44 (IS)
Part 44: Integrated Resources: Product Structure Configuration

[STEP-45] ISO 10303-45 (DIS)
Part 45: Integrated Resources: Material Products

[STEP-203] ISO 10303-203 (IS)
Part 203: Application Protocol: Configuration Controlled Design

Note: the edition of the documents is given between parentheses, using the following abbreviations:

CD	Committee Draft
CDB	Committee Draft, in Ballot
CDBE	Committee Draft, Ballot Ended
CDC	Committee Draft, for Comment
CDCE	Committee Draft, Comments Ended
DIS	Draft International Standard
IS	International Standard

M.2 General Reference Documents

[RFQ] ESA's Request for Quotation "Network Results Format",RFQ/3-8654/95/NL/FG, 11 January 1996.

[SoW] ESA's Statement of Work "Network Results Format" YCV/1746.PPA , Version 2.1,2 October 1995.

[PSS-05-0] ESA Software Engineering Standards, ESA PSS-05-0, Issue 2, 1991

[YCV-09] YCV Standards and Guidelines for Software Documentation ESA/ESTEC/YCV, YCV-09, Version 5.1, 11 April 1989

[YCV-10] Standards and Guidelines for Programs written in C ESA/ESTEC/YCV, YCV-10, Version 2.1, 4 June 1989

[YCV-35] YCV Motif Style Guide

[ESABASE-Lib] ESABASE/LIB-PM-058, Result Data Library Programmer's Manual

[SET-ATS] SET Application Protocol: Data for Spatial Thermal Analyses, GDT-N/93-002, Version 2.2

[GFF] GRABE File Format: GFF description, YCV-023

[TDH] TDH acquisition and processing system, YTO/DES/TDH/0333/C (section 11.1)

[TFCC] Thermal Test Data Format Working Group Report, YTO/REP/TDH/0807

[Dynaworks] Dynaworks V3.1 : Thermal Data Base
Thermal schema 2.0 Reference Guide 4 June 1996
Intespace Toulouse France

[ESATAN-UM] ESATAN User Manual, ESATAN/FHTS version 7 (section 6.2.3)

[CDF] CDF User's Guide, version 2.5, January 1995, National Space Science Data Center

[netCDF] netCDF User's Guide, version 2.3, April 1993, Unidata Program Center

[HDF] HDF Reference Manual, version 3.3, February 1994, National Center for Supercomputing Applications

[GEM-Core] The GEM Core Model
Generic Engineering analysis Model ESPRIT project
Document D2401 revision 4 dated 17 January 1996

[Webster] Hypertext Webster Interface
http://gs213.sp.cs.cmu.edu/prog/webster

M.3 STEP-TAS related documents

[STEP-TAS-AP] STEP Application Protocol – Thermal Analysis for Space
STEP-TAS-012-AP, Version 2, Release 2.0

[STEP-TAS-paper] Hans Peter de Koning and Pau Planas Almazan,
STEP Application Protocol – Thermal Analysis for Space,
SAE Technical Paper Series No. 951725,
25th International Conference on Environmental Systems,
San Diego, CA, USA, July 10-13, 1995.

[STEP-TAS-FR] Final Report Thermal Neutral Format Based on STEP,
STEP-TAS-052-FR, Release 1.0, 31 October 1995.

[ICETAS-URD] ICETAS User Requirements Document,
URD-ICETAS-008, Release 2.2, 18 February 1994.

[ICETAS-SID] ESA ICETAS Standards Document,
SID-ICETAS-009, Release 2.2, 28 January 1994.

[ICETAS-FR] Final Report ICETAS Phase 2,
FR-ICETAS-061, Release 1.1, 19 April 1994.